MAX-PLANCK-INSTITUT FÜR INFORMATIK

Workshop on

Theorem Proving with Analytic Tableaux and Related Methods

Marseille, France, April 28-30, 1993

David Basin, Reiner Hähnle, Bertram Fronhöfer,
Joachim Posegga, Camilla Schwind

MPI–I–93–213 March 1993

Im Stadtwald
66123 Saarbrücken
Germany

Workshop on

Theorem Proving with Analytic Tableaux and Related Methods

Marseille, France, April 28-30, 1993

David Basin, Reiner Hähnle, Bertram Fronhöfer,
Joachim Posegga, Camilla Schwind

MPI–I–93–213 March 1993

Introduction

This report contains most of the 30 abstracts and papers that were accepted to the *Second Workshop on Theorem Proving with Analytic Tableaux and Related Methods* which took place in Marseille France, April 28 – 30, 1993. The first workshop was held in Germany in Lautenbach near Karlsruhe the year before. In contrast to the first workshop, the submitted papers were refereed, since only about half of the contributions could be fit into a three day's workshop.

The workshop was organized by David Basin, Bertram Fronhöfer, Reiner Hähnle, Joachim Posegga, and Camilla Schwind. Its aim was to bring together researchers and groups interested in the mechanization of reasoning with tableaux and related systems. This includes analytic tableaux, model elimination, connection method, and matings. The focus was both on theoretical work and the presentation of implementation techniques and related practical experience. Invited talks were given by Melvin Fitting and Mark Stickel.

The organizers thank the members of program committee which (in addition to the organizing committee) consisted of Masayuki Fujita, Thomas Käufl, Don Loveland, Neil Murray, Peter Schmitt, and Graham Wrightson. Papers were also reviewed by the additional referees, listed below.

> Bernhard Beckert, Philippe Besnard, Christoph Brzoska, Dolors Costal, Jürgen Dix, Christian Fermüller, Melvin Fitting, Harald Ganzinger, Stefan Gerberding, Ch. Goller, Michael Hanus, Andreas Herzig, Hans Juergen Ohlbach, Pascale Kuhna, Hirofumi Kumeno, Bertram Ludaescher, Sean Matthews, K. Mayr, Daniel Mey, Guido Moerkotte, Max Moser, Nicola Olivetti, Sven Eric Panitz, Wolfgang Reif, Vincent Risch, Karl Schlechta, Manfred Schmidt-Schauß, Klaus Schneider, J. Schumann, Irene Stahl, Geoff Sutcliff, Ch. Suttner, Andreas Toenne, Andreas Werner, and Richard Zack.

Support for the workshop was provided by Max-Planck-Institut für Informatik Saarbrücken, Gesellschaft für Informatik, and Centre National de la Recherche Scientifique France; we thank these sponsors. We also thank Pascale Kuhha, Solange Panattoni, and Vincent Vialard from the Groupe Intelligence Artificielle for their help with local arrangements.

Contents

METEOR: Exploring Model Elimination Theorem Proving

Owen Astrachan
Department of Computer Science
Duke University
Durham, NC USA
ola@cs.duke.edu

Abstract

In this paper we describe the theorem prover *METEOR* which is a high-performance Model Elimination prover running in sequential, parallel and distributed computing environments. *METEOR* has a very high inference rate, but as is the case with better chess-playing programs speed alone is not sufficient when exploring large search spaces; intelligent search is necessary. We describe modifications to traditional iterative deepening search mechanisms whose implementation in *METEOR* result in performance improvements of several orders of magnitude and that have permitted the discovery of proofs unobtainable by top-down Model Elimination provers.

1 Introduction

Model Elimination (*ME*) [Lov68, Lov69, Lov78] is the basis for the underlying inference mechanism of several high-performance theorem provers. The design of these provers is adapted from the architecture of the WAM (*Warren Abstract Machine*) [War83] — the *de facto* standard for efficient Prolog implementations. Such provers include *PTTP* (*Prolog Technology Theorem Prover*) [Sti86, Sti88], the prover *SETHEO* [LBSB92] and its parallel counterpart *PARTHEO* [SL90], the *PARTHENON* prover [BCLM92], and the *METEOR* family of provers [AL91, AS92, Ast92].

Although the parallel implementations of *ME* may exhibit high rates of inference, exploration of the large search spaces typical of "difficult" theorems demands an approach other than fast brute-force search using pure Model Elimination. In this paper we provide performance measurements for the *METEOR* family of provers that indicate the inference rate of the prover in its parallel and distributed versions is as great or greater than the other aforementioned provers. We also describe modifications to the search mechanism that permit the discovery of proofs that are infeasible to obtain without such modifications. The descriptions and measurements are taken from [Ast92] which extends work previously reported in [AL91] (concerning the parallel prover) and in [AS91, AS92] (concerning modifications to the search mechanisms).

This paper is organized as follows: In Section 2 we give a very brief description of the *METEOR* architecture. Familiarity with the *ME* *extension* and *reduction* inference rules and other *ME* terminology is assumed in this section and throughout this paper, see [Lov78] for details. In Section 3 we provide performance measurements obtained from the parallel and distributed versions of *METEOR*. In Section 5 we outline *caching* and *lemmaizing*, our modifications to the search mechanism that permit the discovery of difficult theorems; in this section we also provide performance measurements emphasizing in particular the promise of lemmaizing. Conclusions are presented in Section 6.

2 The *METEOR* Architecture

METEOR is written in C and runs on machines supporting Unix[1] and its variants (e.g., MACH). The architecture is based on the WAM in that traditional WAM data structures such as the heap (global stack) and trail are maintained. Rather than compiling clauses into WAM-like code, however, clauses are compiled into a data structure that is then "interpreted" by engine(s) running on either a uniprocessor, an MIMD multiprocessor, or a network of heterogeneous workstations. The same data structure is used in all these computing environments which permits *METEOR* to be ported (in theory) to a large class of machines.

The architecture used in *METEOR* is designed to permit efficient sequential execution while allowing *METEOR* to run in both shared memory and message-passing parallel and distributed environments. Only OR-parallelism is supported in *METEOR*. In particular, alternative candidates for the *ME* extension operation provide the means by which parallelism is achieved; reduction alternatives must be explored by the sequential inference engines and do not contribute to OR-parallelism.

METEOR employs a technique wherein the state of a deduction is re-created when "stealing" work as is the case in the *PARTHEO* prover and in the MUSE OR-parallel Prolog system [AK90]. This technique enables *METEOR* to be used in a message-passing environment and in a shared memory environment. The data structure that represents the compiled clauses for a particular theorem makes state re-creation a very efficient operation and does not incur a performance penalty when used for sequential execution. This should not be construed to mean that the sequential version of *METEOR* also employs state re-creation. The sequential version of *METEOR* uses typical Prolog-style backtracking whereby the trail is used to record (and undo) variable bindings. The parallel and distributed versions of *METEOR* also employ this technique except when stealing work from other processors (or the centralized server in the distributed version — see Section 2.2) in which case state re-creation is used.

2.1 Sequential *METEOR*

As in other *ME*-based provers, *METEOR* represents each n-literal clause as n contrapositives[2]. Clauses are indexed on the first argument as in many Prolog implementations in order to reduce the number of extension alternatives.

The tagged memory used in *METEOR* is typical of WAM-based architectures (e.g., *SETHEO* employs a similar tagged memory). Tags are used, for example, to differentiate between constants, terms, and variables. A variable is further distinguished as to whether it is the first occurrence in a clause in which case no occurs-check need be performed.

METEOR also employs consecutively bounded depth-first search [ST85] also called iterative deepening [Kor85] in order to avoid the incomplete search that is typical of Prolog implementations. Several different depth measures may be specified at runtime as is the case with the *SETHEO* prover. Several of these are outlined in [AL91] and a complete description may be found in [Ast92].

[1] Unix is a registered trademark of AT&T.

[2] Such representations can be overridden by annotating an input file, e.g., the clause representing symmetry in a theorem involving inequalities: $(X \leq Y) \vee (Y \leq X)$ should not be represented twice. In addition, refinements to *ME* inference mechanisms such as Plaisted's positive refinement [Pla88] and Spencer's foothold format [Spe90] are supported that may reduce the number of potential reduction alternatives at the expense of potentially longer proofs.

2.2 Parallel *METEOR*

The OR-parallel version of *METEOR* currently runs on a Butterfly TC2000. Each node of this 45 node machine is a Motorola (RISC) 88000 processor with sixteen megabytes of memory per processor and a 32 kilobyte cache (16K instruction, 16K data). The TC2000 uses the same multi-stage interconnection network supporting internode communication as does the Butterfly GP1000, [3] but each node is a much more powerful computing engine. In the TC2000 there is a three level memory hierarchy: cache, local memory, and remote memory.

The design used in *METEOR* is based on the notion of a concurrent pool [Man86, KE89], a data structure that permits efficient migration and sharing of dynamic information. In [AL91] we termed our data structure a SASS-pool — for Simultaneous Access of Selectively Shared objects. A pool as defined in [Man86] holds a number of tasks that is potentially much larger than the number of processors. As processors become idle, tasks are shared or migrated among processors. The size of the set of migrated tasks and the protocol that determines from which processor tasks are taken affects the performance of pools as noted in [KE89]. Although an *ME* task is a reduction or an extension — so that there are many more tasks than there are processors — we have chosen a coarser-grained view and treat a choice point as a task. With this view a task becomes a public node from which work may be stolen and a SASS-pool does not migrate tasks, but makes them available using a locking protocol described below. The *METEOR* pool is a global data structure shared among all processors, as such it is a potential bottleneck. However, results indicate that a very small percentage of runtime is spent in attempting to acquire the locks used to guard access to the pool. The SASS-pool uses a concurrent-read, exclusive-write (CREW) locking protocol as described in [Lam77]. Processors attempting to steal work are *reading* processors whereas the processor from which work is stolen is the *writing* processor. There is no limit as to the number of reading processors that may obtain a lock on a choice point. However, if a choice point is locked by a writing processor (in *METEOR* this is the processor that created the choice point), then reading processors are excluded, i.e., read locks cannot be obtained. A write lock is obtained only when all extension alternatives have been explored and a processor must place another choice point (whose extension alternatives may not be exhausted) in the SASS-pool.

2.3 Distributed *METEOR*

The architecture used in the distributed version of *METEOR* is illustrated in Figure 1. In general, the distributed configuration consists of a *METEOR server* process and one or more *engines* which serve as virtual processors. It is intended that each engine execute on a separate machine, but the user of the distributed system places the engines on machines at startup (using a shell-script). One process is identified as the *master* engine. Processes communicate by sending messages using Unix sockets.

A distributed run of *METEOR* entails the following sequence of steps.

1. The server process is invoked for a specific number of *METEOR* engines and opens a temporary *universal* socket.

2. Each *METEOR* engine connects to the server through the universal socket. Each engine creates a process-unique socket used for communication with the server. The socket-id is sent as a message using the universal socket. Subsequent process-server communication uses the process-unique socket.

[3] *METEOR* was originally developed on the GP1000 and performance data given in [AL91] is from the version of *METEOR* running on a GP1000.

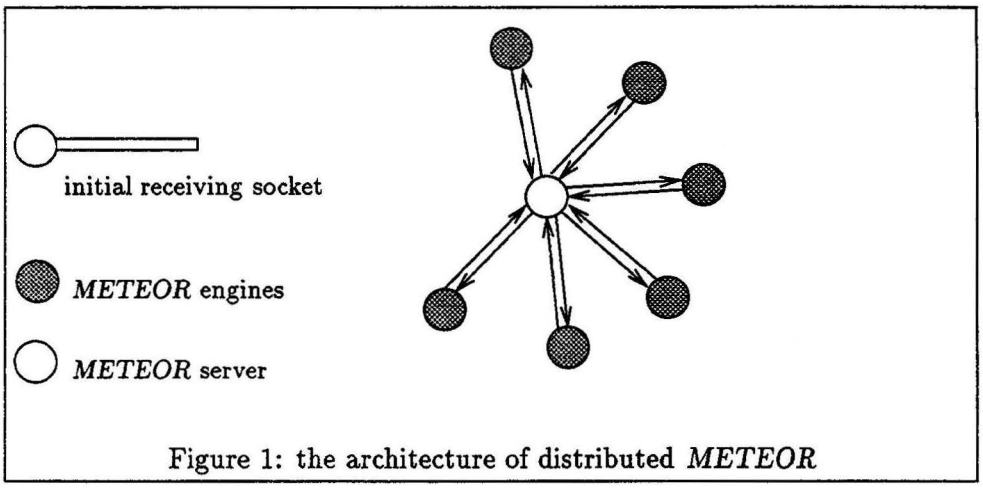


Figure 1: the architecture of distributed *METEOR*

3. Each *METEOR* engine reads the input file and compiles the clauses into a data structure in precisely the same manner as the sequential version of *METEOR*.

4. The master engine begins the search for a proof and shares some work by sending the information used to re-create the work to the server. Non-master engines query the server process for work. When work is received, an engine re-creates the sequence of deductions and begins a parallel search. Work is donated to the server in a manner described below.

5. If all processes are in a query state (see Figure 2), a new depth bound is necessary and the sequence of steps begins anew with the master engine starting the search. If one process finds a proof, the server is notified and the server, in turn, notifies all *METEOR* engines.

6. Each engine shuts down after transmitting statistics to the server and receiving an acknowledgment.

The master *METEOR* engine begins work in a *must-share* mode. In this mode an engine sends a task to the server process for subsequent distribution to engines in need of work. Unlike the tasks in a parallel setting (in which shared tasks were equated with shared or-nodes), a shared task in the distributed setting is one possible extension as specified by the sequence of steps needed to regenerate the state associated with the shared chain. When work is shared, an engine changes from must-share mode to *work* mode in which the engine functions as a sequential processor. When an engine completely explores its search space, it enters *query* mode by sending a request for work to the server process. Initially all engines other than the master engine are in query mode. If all engines are in query mode the server determines that a new level of iterative deepening search is required and the process described above begins anew. If work is available, the server sends a task to the querying engine. If the task sent is generated by engine e, and is the last task stored by the server originating from e, the server sends a message to e indicating that e is to enter must-share mode. In this way the server always has a task or expects to receive one from each processor. A command line option when the engines are started can specify the granularity of task-sharing in the sense that each engine will send n tasks to the server when in must-share mode. The default is to send one task, we use this granularity in the results given below.

When the server receives a work query that cannot be immediately satisfied the query is temporarily queued until tasks are received or until a new stage of iterative deepening

search is started. Queued queries are processed in FIFO order. An engine waiting for a query to be processed is in *idle* mode. A possible optimization involves letting idle engines do speculative work until outstanding must-share requests from the server are answered and re-routed to idle engines. It would be a simple matter, for example, for an idle engine to increase the depth bound that caused its search space to be exhaustively explored with failure and to re-explore the same search space with the higher bound. Although this may result in redundant computations from a global viewpoint, speculative work may result in shorter runtimes.

Transitions between modes in distributed *METEOR* engines are summarized in Figure 2.

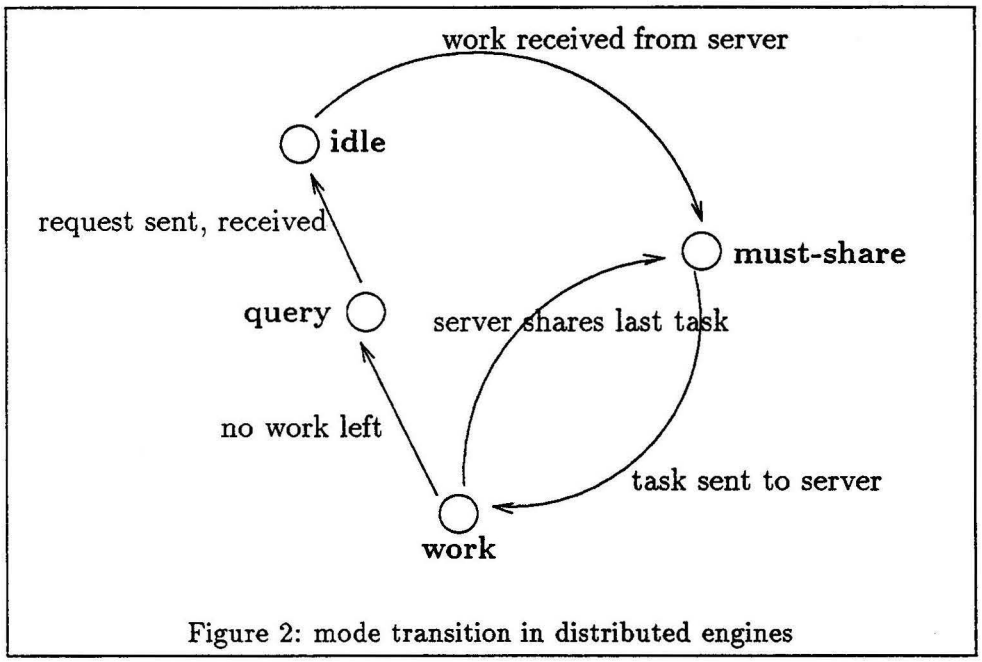


Figure 2: mode transition in distributed engines

3 Parallel and Distributed Performance

3.1 Performance of Parallel *METEOR*

In Table 1 results are given for the runtimes of several benchmark theorems taken from [WM76] and subsequently cited in [Sti88]. The inference rates associated with these theorems are given in Table 2. [4] These theorems are almost all the theorems in the intersection of problems for which data are given in published references for the parallel provers *PARTHEO* and *PARTHENON* (data for the theorems known as *apabhp* and *has-parts2* are also given for *PARTHENON*, several other theorems can be found in *PARTHEO* references). No true comparisons of *METEOR* data with the data from these other provers can be made because of the different computers on which the provers run and because of different depth measures employed. The default depth measure used in iterative deepening with *SETHEO* is the depth of the tree that forms the search space (this is called D_{Alit} in [AS92, Ast92] — it corresponds to the number of A-literals on any one branch of the search tree, and

[4] Statistics given in this section are for the run with minimal runtime taken from a set of five runs of each theorem. In [Ast92] a full discussion can be found as to why runtimes often vary widely across several runs of the same theorem with the same number of processors and how performance results should be made to accurately judge the performance of a parallel prover.

is called *dbs* in [LBSB92]) whereas the default measure used with *PARTHENON* and *METEOR* is the number of inferences in a (potential) proof (this is called D_{inf} in [AS92, Ast92] and *ibs* in [LBSB92] — for inference bounded search). For example, the theorems *wos1* and *wos4* appear to be solved much more rapidly by *PARTHEO* than do other theorems. These theorems are proved much more quickly when D_{Alit} is used whereas the other theorems are proved more quickly when D_{inf} is used. Data using both depth measures with *SETHEO* can be found in [LBSB92] and with *METEOR* in [Ast92]. Finally, the *PARTHEO* prover does not employ true iterative deepening. A depth bound sufficient for discovery of a proof is used in the runs for which data is given in [SL90] and reproduced below. Both *PARTHENON* and *METEOR* employ true iterative deepening search.

# procs	theorem					
	ls36	wos10	wos1	wos22	wos21	wos4
1	343.2	42.54	59.57	43571	1210	2984.4
2	195.09	40.99	31.01	28022	733.8	622.79
5	41.52	19.97	14.73	14808	249.39	101.2
10	22.19	7.26	8.38	8139	190.79	51.28
15	15.71	3.82	7.82	6638	132.35	38.1
20	12.59	2.98	6.3	6309	92.42	34.54
25	10.87	3.89	5.51	6312	69.17	34.07
30	9.88	2.78	6.69	6268	53.23	33.66
35	9.86	2.83	5.23	4494	43.29	29.01
PARTHENON on Encore Multimax						
15	151	20	12	na	373	501
PARTHEO on Transputer T800						
16	352	135.6	1.1	na	na	1.3

Table 1: parallel runtimes for benchmark problems

A complete discussion of why runtimes may be valuable indications of absolute performance but poor for determining the speedup performance of a parallel system can be found in [Ast92]. Note that the inference rate varies less across the set of theorems for which data is given in Table 2 than do the runtimes given in Table 1. In particular one might roughly categorize *METEOR* running on 35 processors of the Butterfly TC2000 as a 30,000 lips prover (note that the rate varies from 25,171 to 61,326 across the set of theorems).

# procs	theorem					
	ls36	wos10	wos1	wos22	wos21	wos4
1	2463	2095	2398	1896	2145	1723
2	4237	3513	4277	3234	3650	3234
5	10393	9126	10355	7853	9043	7629
10	20710	17325	20682	15676	17714	14554
15	30788	28477	30537	23069	25862	21145
20	40633	36129	39907	29109	32250	24032
25	49222	44145	47213	32468	31405	24539
30	57047	50044	52368	32729	36283	24356
35	61326	57395	53889	33066	36623	25171
PARTHENON on Encore Multimax						
15	8197	8419	9465	na	10023	9532

Table 2: parallel inference rates for benchmark problems

4 Performance of Distributed *METEOR*

In Table 3 results are given for the distributed version of *METEOR*. Although figures are given for the mean, mode, and maximal runtimes, only the maximal runtime is a reliable indicator of performance. In particular, running a prover on one processor for 10 seconds and NOT running it on nine other processors yields a mean runtime of one second — hardly an accurate measure of performance. The inference rates associated with the maximal runtimes are given in Table 4.

procs	*mean, mode, max runtime* theorem				
	ls36	wos10	wos1	wos22	wos21
1	208.17	25.63	37.78	22565	704
2	101.51	10.7	17.22	14163	541.82
	101.50	10.27	17.22	14163	541.82
	102.49	10.94	18.27	16874	546.70
5	19.78	6.67	6.26	6521	161.35
	20.90	6.63	6.13	6561	179.32
	21.72	8.15	8.16	6585	185.36
10	11.03	3.66	3.43	4454	51.17
	11.13	3.38	3.31	4680	53.85
	13.58	5.77	5.47	5259	57.35
15	7.61	2.61	2.24	3402	26.67
	7.84	2.54	2.22	3600	28.33
	9.37	5.15	4.01	4371	32.83
20	5.83	2.11	0.88	2975	17.50†
	5.76	1.99	1.51	2931	16.85
	8.48	4.59	3.46	3054	34.74

†25 engines

Table 3: runtimes using distributed *METEOR*

procs	*max lips rate* theorem				
	ls36	wos10	wos1	wos22	wos21
2	6605	5947	6528	5096	7086
5	16327	14629	15001	15340	15137
10	28111	21605	22344	25781	32540
15	43374	25989	31004	36350	44882
20	50176	33952	35727	58330	40360†

†25 engines

Table 4: lips rate using distributed *METEOR*

Note that using 20 workstations yields generally higher inference rates and better overall performance than using 30 processors of the shared memory TC2000.

5 Caching and Lemmaizing

Caching and lemmaizing are used, respectively, to replace and augment the normal search mechanism. A brief description of the differences between these two modifications to the

search procedure is given below, a more complete development can be found in [AS91, AS92]. In our implementation caching is only applicable to theorems expressible as Horn clause sets. Lemmaizing, however, can be used with both Horn and non-Horn theorems.

In both hardware caches and memoizing software caches the cache incurs an increase in storage costs in order to reduce some other resource such as runtime or disk accesses. In *METEOR* caching refers to a methodology that is a hybrid of these memoizing methods. We use an idea of Stickel's whereby caching is not used to augment the normal searching mechanism, but to replace it. In our terminology, *caching* refers to a mechanism that optionally replaces the normal search mechanism with a cache that has a lower computational cost, but yields essentially identical results to search. The objective of caching is to make effective use of results discovered in past search. Caching simply stores results of past searches and replaces future searches by cache lookup. Proofs will be found with the same cost bound as when caching is not used, and no more inferences will be performed with caching than without (except that some pruning mechanisms must be disabled when caching is employed, e.g., the identical-ancestor pruning rule [Sti88, AS92]). Whether there is a net performance gain depends on how efficiently the cache is implemented, i.e., what are the relative costs of search and cache lookup.

Caching is intended to function as a redundancy reducing mechanism in that it allows some subsumption tests to be incorporated into backward-chaining reasoning systems Although heuristic caching [5] does permit the discovery of proofs at depths less than would be necessary without such caching, neither caching nor heuristic caching permits previously proved solutions to be used as lemmas in the manner that mathematicians use lemmas to shorten the presentation and development of what might otherwise be unwieldy proofs. We use *lemmaizing* to refer to a method in which only certain (it is hoped relevant) solutions are stored (rather than all solutions); these solutions are used to augment the normal search mechanism. By identifying a relevant solution whose derivation may require many inference steps, but not deducting the cost of the derivation from the available resources when the stored solution is used, it is possible to discover high-cost proofs with low cost bounds. These relevant solutions are treated as input unit clauses and used in extension operations thus augmenting rather than replacing search. We reserve the word caching for use with the mechanism in which cache lookups replace search.

Since lemmas are not needed for completeness, we may impose arbitrary syntactic and semantic criteria when deciding which lemmas to retain. The idea is to store lemmas that are used to eliminate repeated subdeductions. In this sense the use of lemmas allows us to combine an aspect of bottom-up reasoning with the top-down reasoning in *METEOR*. By imposing strict criteria on lemmas we retain a complete inference system that will hopefully allow us to prove theorems otherwise unobtainable. In one of the first implementations of *ME* [FLSY74], this kind of lemma use was explored in an ad-hoc manner. No notion of lemma cost was explored and lemmas were not found to be useful in general since the potential for a smaller search space was not realized because of the increased branching factor induced by allowing lemmas as alternatives for extension.

In *METEOR* we have implemented a lemmaizing mechanism that permits the discovery of proofs that could not possibly discovered with the normal inference mechanism because of space and time considerations. Our notion of lemmaizing also permits demodulation with lemmas in a complete *ME*-based inference system.

[5] Heuristic caching refers to a method in which all solutions are stored, but a cost other than discovery cost is charged when a cached subgoal is retrieved.

5.1 Performance Using Caching

We will emphasize the role of lemmaizing in this paper, but include some comments on the role of caching for completeness. Full details on caching may be found in [AS92, Ast92]. For problems for which data is given in Table 3 and Table 1, caching significantly reduces the time needed to find a proof. There are problem domains, however, for which storing all solutions as is required when caching is employed does not yield a performance gain and may significantly degrade performance compared to non-caching runs. One domain in which we have not had success employing caching is the set of problems employing the inference rule of condensed detachment reported in [MW92].

We also note the successful use of caching and heuristic caching with the theorem known as SAM's lemma [Wos88]. This theorem is proved in 300 seconds using caching and 37 seconds using heuristic caching (extensive data may be found in [Ast92]). Projecting the results gleaned from the proof found using caching and the branching factor of non-caching runs indicates that more than 10^{14} seconds ($> 3,000,000$ years) would be required to find a proof if neither caching nor lemmaizing is employed.

5.2 Performance Using Lemmaizing

Statistics using lemmaizing with the same benchmark theorems we have used previously are given in Table 5. Both time (in seconds) and inference rate are given for three different methods of employing lemmaizing. The row labeled *lemma* represents the storage and use [6] of all lemmas whose terms meet a maximum term-depth criterion as indicated in the table.

Lemmaizing in *METEOR* can also be used with demodulation. In the current system we use lexicographic recursive path ordering (LRPO) [Der82] based on a user-specified total ordering of symbols to determine if a rewrite rule applies. The user may specify if a set of rewrite rules is to be used in addition to or independently of any dynamically generated demodulators.

The row labeled *lex* in Table 5 gives data for lemmaizing when lemmas based on the equality literal are oriented according to the user-supplied order, but no rewriting occurs. The row labeled *dynamic* represents the use of generated equality lemmas to dynamically rewrite other lemmas. Rewriting can reduce the number of stored lemmas (e.g., by rewriting lemmas that would otherwise be different so that they are syntactically identical) and can result in the storage and use of lemmas that would otherwise not meet the syntactic limitations placed on stored lemmas (by rewriting, for example, a lemma whose terms are nested to a depth of three to a lemma with maximum term-depth one).

Although the use of lemmaizing enables *METEOR* to prove many benchmark theorems more quickly than when lemmaizing is not employed, a better indication of its promise is its use in proving theorems that are otherwise infeasible. For example, the theorems known as *wos20*, *wos26*, *wos28*, and *wos33* cannot be proved in 24 hours of cpu time using any of the provers previously discussed (including *METEOR* when lemmaizing is not employed). All of these theorems have proofs that are found in less than 60 seconds when *METEOR* is used with lemmaizing (and, in some cases, demodulation). The group theory challenge problems known as the *commutator problem* and the *boolean ring problem* [WO91] are also proved using lemmaizing in roughly 355 seconds and 2,300 seconds respectively.

In addition to these problems from group and ring theory, we have employed lemmaizing to prove several theorems from real analysis given in [WB87] and [Ble90]. The problem

[6] A threshold is used when lemmaizing is employed just as when caching is employed. In the case of lemmaizing the use of the threshold is not predicated on the need to dispense with the identical ancestor pruning rule as with caching, but because "shallow" use of lemmas may not be useful given the fast inference rate of *METEOR*. In this section the default threshold is employed.

method	*time (seconds) and inference rate (lips)* theorem					
	ls36	wos10	wos1	wos15	wos21	wos4
lemma	46.9†	4.83†	0.4‡	23.1†	> 1000‡	81.66†
	4,135	4,050	3,807	4,558		3,316
lex	25.5	4.11	0.39	22.22	> 1000	76.1
	4,120	4,036	3,623	4,570		3,261
dynamic	7.45	2.64	0.36	3.74	23.99	53.75
	4,390	4,120	3,700	4,811	2,831	3,850

†termdepth limit 1 ‡termdepth limit 2

Table 5: lemmaizing and demodulation for benchmark theorems

known as AM8 (a continuous function attains a maximum on a closed interval) is proved in less than one minute, the intermediate value theorem (IVT) is proved in approximately 10 seconds. Finally, the third in a sequence of five problems regarding the continuity of the sum of continuous functions is provable in approximately 7 hours (the first two formulations are easily proved using one of the modified depth measures employed in *METEOR* and first reported in [AL91]).

Data in Table 6 indicate the performance of *METEOR* when lemmaizing is employed. The entry for stored lemmas indicates the number of lemmas meeting the term-size and term-depth limits as noted. The number of generated lemmas includes lemmas that fail to meet the size constraints and that meet the constraints but are subsumed by lemmas already stored.

Note that the term-size limit employed in the real analysis problems AM8 and IVT are robust in the sense that performance degrades gracefully and proofs are found as the term-size limit is increased. There is a minimum term-size below which lemmas necessary for proofs are not generated. Complete statistics for these problems are given in [Ast92]. The data for the group and ring problems do not necessarily represent the best runtimes for these problems, but do represent the use of default search strategies in *METEOR*. For example, if user supplied demodulators are used in addition to dynamically generated demodulators the runtimes often decrease significantly. Complete statistics for these problems are also given in [Ast92].

theorem	time (secs)	lips	# lemmas stored	# lemmas gen. (10^4)
wos33†	30.45	4,285	137	3.06
wos26‡	16.76	3,370	134	3.16
wos28‡	40.67	2,114	92	8.77
wos20‡	17.82	4,089	52	8.12
commutator‡	487	3,647	432	136.37
boolean‡	2,362	3,285	183	682.84
AM8§	55.78	2,668	208	7.32
IVT§	10.74	3,618	45	0.153

†lex ordering only, term-depth limit 1
‡dynamic demod, term-depth limit 1
§term-size limit 9

Table 6: statistics for lemmaizing with difficult problems

6 Conclusion

We have provided descriptions and statistics for the Model Elimination theorem prover *METEOR* that indicate its promise as a prover and as a means for experimenting with and developing modifications for search procedures and depth measures associated with iterative deepening search in the context of theorem proving.

Data indicate that the parallel and distributed versions of *METEOR* have very high inference rates, but that difficult theorems do not succumb to inference rates alone. We have outlined modifications to the search mechanisms employed in *METEOR* called caching and lemmaizing which permit the generation of proofs for theorems whose solutions would be infeasible with brute-force search mechanisms.

Future work includes experimenting with semantic criteria for constraining the use of lemmas; our work to date has employed syntactic criteria almost exclusively. Using lemmaizing in conjunction with the distributed version of *METEOR* may yield new and improved results as well.

Acknowledgments

This work was done in collaboration with Mark Stickel and Don Loveland who have contributed significantly to the work reported here.

References

[AK90] K.A.M. Ali and R. Karlsson. The Muse or-parallel Prolog model and its performance. In S. Debray and M. Hermenegildo, editors, *Proceedings of the North American Conference on Logic Programming*, pages 757–776, 1990.

[AL91] O.L. Astrachan and D.W. Loveland. METEORs: High performance theorem provers using model elimination. In R.S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Kluwer Academic Publishers, 1991.

[AS91] O.L Astrachan and M.E. Stickel. Caching and lemmaizing in model elimination theorem provers. Technical Report 513, SRI International, Menlo Park, CA, December 1991.

[AS92] O.L. Astrachan and M.E. Stickel. Caching and lemmaizing in model elimination theorem provers. In Deepak Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*. Springer Verlag, 1992.

[Ast92] O.L. Astrachan. *Investigations in Model Elimination Based Theorem Proving*. PhD thesis, Duke University, 1992.

[BCLM92] S. Bose, E. Clarke, D.E. Long, and S. Michaylov. Parthenon: A parallel theorem prover for non-Horn clauses. *Journal of Automated Reasoning*, 8, 1992.

[Ble90] W.W. Bledsoe. Challenge problems in elementary calculus. *Journal of Automated Reasoning*, 6(3):341–359, 1990.

[Der82] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[FLSY74] S. Fleisig, D. Loveland, A. Smiley, and D. Yarmash. An implementation of the model elimination proof procedure. *Journal of the Association for Computing Machinery*, 21:124–139, January 1974.

[KE89] D. Kotz and C.S. Ellis. Evaluation of concurrent pools. In *Proceedings of the Ninth Internation Conference on Distributed Computing Systems*, pages 378–385, 1989.

[Kor85] R.E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[Lam77] L. Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, 1977.

[LBSB92] R. Letz, S. Bayerl, J. Schumann, and W. Bibel. SETHEO—a high-performance theorem prover. *Journal of Automated Reasoning*, 8:183–212, 1992.

[Lov68] D.W. Loveland. Mechanical theorem proving by model elimination. *Journal of the Association for Computing Machinery*, 15(2):236–251, April 1968.

[Lov69] D.W. Loveland. A simplified format for the model elimination procedure. *Journal of the Association for Computing Machinery*, 16(3):349–363, July 1969.

[Lov78] D.W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, 1978.

[Man86] U. Manber. On maintaining dynamic information in a concurrent environment. *SIAM Journal of Computing*, 15(4):1130–1142, 1986.

[MW92] W. McCune and L. Wos. Experiments in automated deduction with condensed detachment. In Deepak Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction.* Springer Verlag, 1992.

[Pla88] D. Plaisted. Non-Horn clause logic programming without contrapositives. *Journal of Automated Reasoning*, 4(3):287–325, September 1988.

[SL90] J. Schumann and R. Letz. PARTHEO: A high performance parallel theorem prover. In *Proceedings of the Tenth International Conference on Automated Deduction*, pages 40–56, 1990.

[Spe90] B. Spencer. Avoiding duplicate proofs. In S. Debray and M. Hermenegildo, editors, *Proceedings of the North American Conference on Logic Programming*, pages 569–584, 1990.

[ST85] M.E. Stickel and W.M. Tyson. An analysis of consecutively bounded depth-first search with applications in automated deduction. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1073–1075, August 1985.

[Sti86] M.E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. In *Proceedings of the Eighth International Conference on Automated Deduction*, pages 573–587. Springer-Verlag, 1986.

[Sti88] M.E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:343–380, 1988.

[War83] D.H.D. Warren. An abstract Prolog instruction set. Technical Report 309, SRI International, Menlo Park, California, October 1983.

[WB87] T.C. Wang and W.W. Bledsoe. Hierarchical deduction. *Journal of Automated Reasoning*, 3:35–77, 1987.

[WM76] G.A. Wilson and J. Minker. Resolution, refinements, and search strategies: A comparative study. *IEEE Transactions on Computers*, C-25(8):782–801, August 1976.

[WO91] L. Wos and R. Overbeek. Subsumption, a sometimes undervalued procedure. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson.* MIT Press, 1991.

[Wos88] L. Wos. *Automated Reasoning: 33 Basic Research Problems.* Prentice Hall, 1988.

Combining Model Elimination with Unit-Resulting Resolution

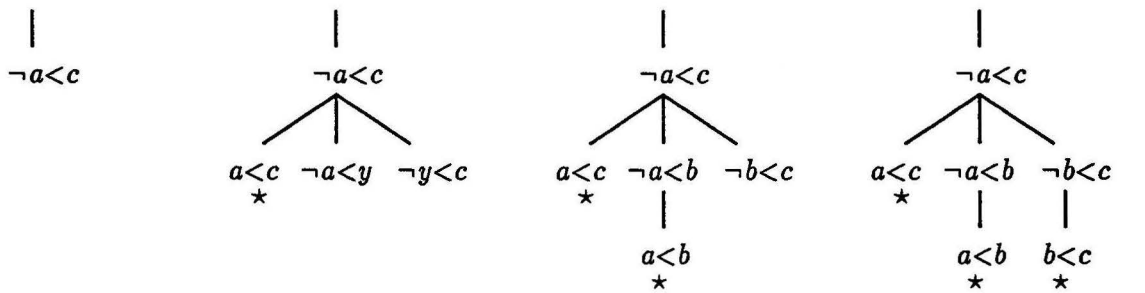
Peter Baumgartner
Universität Koblenz
Institut für Informatik
Rheinau 3-4
5400 Koblenz
E-mail: peter@infko.uni-koblenz.de

In this abstract, we will sketch a refinement of the model elimination calculus (Lov68; Lov78) that allows for significant performance improvements. The idea is to separate away a part of the given clause set and treat this part by a more efficient inference rule than the standard extension inference rule. Here we will consider the unit-resulting resolution inference rule (UR-resolution, see (MOW76)) as such a rule. This technique behaves much like building in the theory of equality by a paramodulation inference rule — however for more general theories and in a general way.

In a first step (section 1) we will extend the model elimination calculus with the inference rule "unit-resulting resolution step" (UR-step). If no further precautions are taken, the naive separation of some input clauses and treating them by UR-steps in general destroys completeness. Hence completeness has to be recovered in some way. Here we can only indicate how this is done, namely by a new technique called "linear completion", which is a saturation process among the separated clauses (section 2). Section 3 briefly describes an implementation, and contains results from practical experiments.

1. Model Elimination with UR-Steps. We will follow (BF92; LSBB92) and define the inference rules of model elimination as tree-transforming operators. Then model elimination can be seen as a restriction of semantic tableaux with unification for clauses (see (Fit90)). This restriction means that a β-step is permissible only if at least one of the resulting new leafs is complementary to its father node.

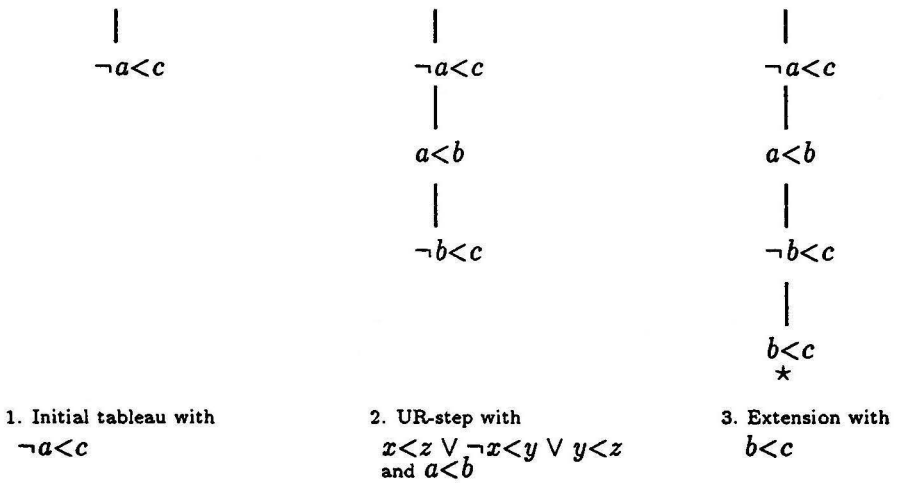
The following example will help to explain the difference to the modified calculus below. Let $M = \{\neg a < c, \quad a < b, \quad b < c, \quad x < z \vee \neg x < y \vee \neg y < z\}$ be a clause set. Consider the following model elimination refutation of M:



1. Initial tableau with $\neg a<c$ | 2. Extension step with $x<z \vee \neg x<y \vee \neg y<z$ | 3. Extension step with $a<b$ | 4. Extension step with $b<c$

The "critical" clause in this example is the transitivity clause $x < z \vee \neg x < y \vee \neg y < z$: instead of extension with $a < b$ in step 3 it were also possible to extend the tree 2 with

ever new instances of the transitivity clause in an infinite way. In order to avoid such unproductive processing, we propose to extend model elimination with a new inference rule *unit-resulting resolution step (UR-step):* in unit-resulting resolution a resolvent from an n-literal clause and $n-1$ or n literals is built by simultaneously resolving upon the $n-1$ or n literals against the n-literal clause. Thus the resolvent is either a single literal or the empty clause. In other words, non-unit clauses only serve as generators for unit clauses. This strategy can be brought into model elimination in the following way: suppose we want to extend the leaf $\neg a<c$ (the initial tree above) in the UR-style with the transitivity clause $x<z \vee \neg x<y \vee \neg y<z$. The leaf $\neg a<c$ can be resolved against the $x<z$ literal, yielding the *intermediate* clause $\neg a<y \vee \neg y<c$. In order to resolve away one further literal of the intermediate clause we have two alternatives: either we can find a suitable literal in the leafs ancestor list, or we can find it in an input clause. Since in the example the ancestor list is empty, we have to chose the latter alternative. A suitable input clause is the unit clause $a<b$. Resolving the intermediate clause with $a<b$ yields $\neg b<c$. This literal plays the role of a new subgoal to be proved; in the viewpoint of *theory reasoning* (Sti85; Bau92) we have executed a partial theory inference step with residue $\neg b<c$. Finally, the model elimination tableau is extended with the used input clauses and the residue as shown in step 2 of the following refutation:



Note that the clause $x<z \vee \neg x<y \vee \neg y<z$ does not explicitly appear in the tree like an input clause; it merely has been turned by means of unit-resulting resolution into an inference rule for transitivity. In summary, UR-steps enable a dedicated treatment of clauses separated away from the input clauses in a unit-resulting way.

2. Linear Completion. Unfortunately, the naive separation of input clauses and treating them with UR-steps leads to incompleteness, even in the Horn case. The reason is, that although UR-resolution is complete for Horn theories, it is not compatible to the linear (i.e. goal-oriented) restriction. This combination, however, is required to obtain the completeness of the overall calculus. In order to repair the situation, the separated clauses have to be preprocessed in such a way that the combination of unit-resulting resolution with the linear restriction *is* complete.

We have designed such a procedure called *linear completion*. It works for a wide class of theories, namely Horn theories (note: *not* definite theories). Thus e.g. the theories of equality, strict and partial orderings, equational theories (e.g. associativity) and their combinations are covered.

The term "completion" can be seen in analogy with Knuth-Bendix completion: Knuth-Bendix completion relates to equational theories and ordered derivations as our completion relates to general Horn theories and linear derivations. As Knuth-Bendix completion, our completion can be understood as a sort of compiler, which compiles a specification once and for all into an efficient algorithm. This process may terminate, at best in hopefully many interesting cases. As an example for a finite system we can mention a system for the theory $\mathcal{ES}$ of strict orderings together with an equivalence relation. If the completion does not terminate, we arrive at an "unfailing" procedure, i.e. for every provable goal eventually enough inference rules will generated that are sufficient to prove the goal. Example: if we add the substitution axioms to $\mathcal{ES}$ we arrive at a system for full equality and strict orderings. Dedicated inference rules for orderings are described in (MW92; Hin92).

For the presentation of linear completion we find it convenient to introduce *UR-rules:* An UR-rule is an expressions of the form $L_1, \ldots, L_n \rightarrow L_{n+1}$, where all $L_1, \ldots, L_n$ are literals (called *premise literals*), and L_{n+1} is either a literal or *false* (called *conclusion*). The declarative meaning of an UR-rule is the implication $(L_1 \wedge \ldots \wedge L_n) \rightarrow L_{n+1}$. Now, unit-resulting resolution derivations can be defined on top inferences with UR-rules, which are in turn defined as follows (ground case): given a rule $L_1, \ldots, L_n \rightarrow L_{n+1}$ and n unit clauses $L_1, \ldots, L_n$. Then L_{n+1} may be inferred. Rules with $L_{n+1} = \mathit{false}$ are used to terminate a derivation and indicate that a proof has been found.

Linear completion works as follows: the initially given set of Horn clauses H is rewritten as a set of UR-rules R in the following way: (1) every unit clause $L \in H$ becomes $\overline{L} \rightarrow \mathit{false}$ in R, (2) every non-unit clause $L_1 \vee \ldots L_n \vee L_{n+1}$ becomes $\overline{L}_1, \ldots, \overline{L}_n \rightarrow L_{n+1}$, and (3) for every predicate symbol p occuring in H the UR-rule $p(\vec{x}), \neg p(\vec{x}) \rightarrow \mathit{false}$ is added. This initial rule set is stepwisely transformed by *deduction rules* into new rule sets. The central deduction rule is the "critical pair rule" (termed in analogy to critical pairs in Knuth-Bendix completion) that allows to obtain a new UR-rule from two present UR-rules by "chaining": consider e.g. the rules $B \rightarrow C$ and $A, C \rightarrow \mathit{false}$. Since we conclusion of the first rule, C, is contained (in the first order case unification is used) in the a premise literal of the second rule, we can generate a new UR-rule in the following way:

$$\frac{\begin{array}{lll} B \rightarrow C & & \\ C & , \; A \rightarrow \mathit{false} & \end{array}}{B \quad , \; A \rightarrow \mathit{false}}$$

The purpose of the newly generated rule is to detour violations of the linearity property in derivations caused by a combination of the rules $B \rightarrow C$ and $B, C, A \rightarrow \mathit{false}$.

Besides the critical-pair rule, at least a deduction rule is needed that means, in resolution terminology, the unit resolution among unit UR-rules (rules of the form $L \rightarrow \mathit{false}$) and non-unit UR-rules. Finally, a redundancy criterion is needed in practice which allows to delete UR-rules. An UR-rule is redundant iff its use in a derivation can be simulated by a smaller derivation (wrt. some noetherian ordering on derivations) using other UR-rules from S.

If all these deduction rules are applied in a *fair* way, completeness of the resulting system wrt. the unit-resulting and linearity properties is guaranteed. Such a system then can be used in a complete combination with model elimination.

3. A PTTP-implementation. We have implemented a PTTP ("Prolog Technology Theorem Prover") (Sti88) in Prolog, extended it towards UR-model elimination and run several benchmarks. The completion of the separated clauses, however, was carried out by hand and stopped after sufficiently many rules for the problem at hand were generated.

Figure (1) contains the runtime results for SEPIA Prolog running on a SPARC ELC. The often dramatical speedups suggest to us that UR-model elimination when combined with linear completion is a significant technique. It appears to us that the more clauses are separated and treated by linear completion, the better the results are. We are currently investigating more test problems in order to support this claim.

Problem	PTTP	*PTTP* with UR-Steps (Separated clauses)
Assoc	12.1	0.28 (Equality+Assoc)
Graph	3219	0.26 (Trans+Sym)
Wos4	89	1.2 (Equivalence+Psub)
Pell48	>3000	0.3 (Equivalence)
Verification	>3000	3.9 (Equality+<+Succ)

Figure 1: Runtime results in seconds. The Problems are: *Assoc:* Unification of two deeply nested terms modulo associativity; *Graph:* Searching an And-Or Graph with transitive and symetric reachability relation; *Wos4:* group theory; *Pell48:* Pelletier 48; *Verification:* Proving the loop invariant of an imperative program to compute an array's maximum element, with equality, strict ordering and succesor arithmetic treated by UR-steps.

References

(Bau92) P. Baumgartner. A Model Elimination Calculus with Built-in Theories. In H.-J. Ohlbach, editor, *Proceedings of the 16-th German AI-Conference (GWAI-92)*, pages 30–42. Springer, 1992. LNAI 671.

(BF92) P. Baumgartner and U. Furbach. Consolution as a Framework for Comparing Calculi. Forschungsbericht 11/92, University of Koblenz, 1992. (submitted to JSC).

(Fit90) M. Fitting. *First Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer, 1990.

(Hin92) L.-M. Hines. The Central Variable Strategy of Str+ve. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE-11)*, pages 35–49. Springer-Verlag, June 1992. LNAI 607.

(Lov68) D. W. Loveland. Mechanical Theorem Proving by Model Elimination. *JACM*, 15(2), 1968.

(Lov78) D. Loveland. *Automated Theorem Proving - A Logical Basis.* North Holland, 1978.

(LSBB92) R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performace Theorem Prover. Journal of Automated Reasoning, 1992.

(MOW76) J. McCharen, R. Overbeek, and L. Wos. Complexity and related enhancements for automated theorem-proving programs. *Computers and Mathematics with Applications*, 2:1–16, 1976.

(MW92) Zohar Manna and Richard Waldinger. The Special-Relation Rules are Incomplete. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE-11)*, pages 492–506. Springer-Verlag, June 1992. LNAI 607.

(Sti85) M.E. Stickel. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

(Sti88) M. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

Extended Abstract

A Completion–Based Method for Adding Equality to Free Variable Semantic Tableaux

Bernhard Beckert

Institute for Logic, Complexity and Deduction Systems
University of Karlsruhe
Am Fasanengarten 5, 7500 Karlsruhe, Germany
beckert@ira.uka.de

1 Introduction

For both classical and nonclassical first–order logic equality is a crucial feature to increase expressivity of the object language. It is, therefore, of great importance to have a method at hand that allows tableau–based theorem provers to handle equality in an efficient way.

In the first approaches to adding equality to semantic tableaux [8, 10, 11] additional tableau rules were defined that allow the application of equalities occurring on a branch to other formulas. In [4] a completion–based method has been described; it uses, however, as the approaches in [8, 10, 11], the ground version of tableaux. Recent methods, described in [5] and [2, 3], are based on the much more efficient free variable tableau system [5].[1]

In [3] it has been shown that one of the reasons for the inefficiency of previous approaches is that they all mix up the application of equalities and of classical tableau rules. In contrast, the method presented in [3] for closing a tableau branch, consists of two separate tableau expansion stages. In the first stage the tableau is expanded using the classical tableau rules; in the second stage, from a branch B the set of equalities[2]

$$E_B := \{s \approx t \ : \ \mathsf{T}(s \approx t) \in B\} \ ,$$

and the set Dis_B are extracted; where the latter consists of disjunctions of inequalities. It is built from the two remaining types of important formulas on B: For every pair $\langle \mathsf{T}\, P(s_1, \ldots, s_n), \mathsf{F}\, P(t_1, \ldots, t_n)\rangle$ of atoms, that potentially close B, in Dis_B there is the n–place disjunction $s_1 \not\approx t_1 \vee \ldots \vee s_n \not\approx t_n$ and for every inequality $\mathsf{F}(s \approx t)$ on B, in Dis_B there is the (one–place) disjunction $s \not\approx t$.

Using these two sets, all substitutions that allow to close a branch B can be generated by computing for each inequality $(s \not\approx t)$ in Dis_B a complete set of rigid E_B–unifiers of s and t:

[1] The γ–rule in free variable tableaux does not substitute the bound variable by a "guessed" ground term, but by a new free variable. Therefore, a free variable tableau can contain equalities that are not ground. To close a free variable tableau T, one has to find a (ground) substitution σ that allows to close all branches of T simultaneously, i.e., such that $T\sigma$ is closed.

[2] We use the signed version of tableaux, i.e., the node labels in a tableau are first–order formulas prefixed with either T (true) or F (false). The equality predicate symbol is denoted by $\approx$, such that no confusion with the meta–level equality predicate $=$ can arise.

Definition 1 (Rigid E–Unifier, Complete Set of Rigid E–Unifiers) *Let E be a finite set of equalities and let s and t be terms. A substitution σ is called a rigid E–unifier of s and t iff*

$$E\sigma \models s\sigma \approx t\sigma$$

where the variables in $E\sigma$, $t\sigma$ and $s\sigma$ are treated as constants (held "rigid"), i.e., $E\sigma \models s\sigma \approx t\sigma$ is treated as a ground problem.

A set $C(E, s \approx t)$ of rigid E–unifiers of s and t is called complete iff for each rigid E–unifier τ of s and t there is a unifier $\sigma \in C(E, s \approx t)$ that is more general[3] than τ.

In general, there is no *finite* complete set of rigid E-unifiers; it is, however, always possible to enumerate one.

Example 2 *The following table shows some examples for rigid E–unification problems:*

E	s	t	Rigid E–unifiers of s and t
$\{f(x) \approx x\}$	$f(a)$	a	$\{x/a\}$
$\{f(a) \approx a\}$	$f(x)$	a	$\{x/a\}, \{x/f(a)\}, \{x/f(f(a))\}, \ldots$
$\{f(x) \approx x\}$	$g(f(a), f(b))$	$g(a, b)$	none[4]

Each rigid E–unifier is as well a non–rigid E–unifier, but, as the last of the above examples shows, the contrary does not hold true.

With complete sets of rigid E–unifiers for each inequality $(s \not\approx t)$ in Dis_B at hand one can easily construct all closing substitutions of the branch B by searching for substitutions τ that are a specialization of some $\sigma_i \in C(E_B, s_i \approx t_i)$ $(i = 1, \ldots, n)$, where $(s_1 \not\approx t_1 \vee \ldots \vee s_n \not\approx t_n)$ is one of the disjunctions in Dis_B. These substitutions τ allow to *simultaneously* prove all the inequalities in one of the disjunctions to be false.[5]

The method described in [3] for computing complete sets of E–unifiers is based on iteratively constructing sets $\langle t \rangle_B$, whose elements t'_σ are terms labeled with a most general substitution that allows to derive them from t using equalities in E_B, i.e., if $t'_\sigma \in \langle t \rangle_B$ and τ is a specialization of σ, then $E\tau \models (t \approx t')\tau$. With that

$$C(E_B, s \approx t) = \{\sigma \;:\; \text{there are } s'_{\sigma_1} \in \langle s \rangle_B,\ t'_{\sigma_2} \in \langle t \rangle_B \text{ such that } s',\ t' \text{ are unifiable with MGU } \sigma_3, \text{ and } \sigma \text{ is a most general specialization of } \sigma_1,\ \sigma_2,\ \sigma_3\ \}$$

is a complete set of rigid E–unifiers of s and t.

Even if this method for handling equality within a first–order analytic tableaux framework outperforms all previous approaches considerably, it cannot compete with completion–based equality provers or resolution systems using paramodulation. The main problem is that for computing the sets $\langle t \rangle_B$ equalities are applied unrestricted in both directions. Therefore, $\langle t \rangle_B$ may contain many redundant terms.

In addition, this and all other procedures that allow equalities to be applied unrestricted in both directions have another major drawback: They cannot be used to *decide* the rigid E–unification problem, i.e., to decide whether a rigid E–unifier of two terms s and t exists, although this problem is, in fact, decidable [6].[6] Therefore, the search might continue infinitely even if no rigid E–unifier exists.

[3] σ is more general than τ iff $\tau = \sigma\mu$ for some substitution μ.

[4] There is, however, a *non*–rigid E–unifier of s and t (the empty substitution).

[5] If τ is a specialization of a rigid E–unifier σ, then τ is a rigid E–unifier as well.

[6] The problem has been shown to be NP–complete.

2 A Completion–Based Method

All disadvantages mentioned above can be avoided by using a completion based procedure for computing complete sets of rigid E–unifiers. The algorithm we use is an extension of the algorithm that Gallier et. al. use in [6] to prove the decidability of the rigid E–unification problem. It is based on the unfailing completion procedure [1, 9]. The basic idea — and the main difference to the classical unfailing completion procedure — is that during the completion process variables are never renamed, even if equalities that have variables in common are applied to each other. In addition, *constraints* $\langle \sigma, O \rangle$ consisting of a substitution σ and a set O of *order conditions*[7] are attached to the reduction rules: $l \to r \ll \langle \sigma, O \rangle$ means that $l\tau \to r\tau$ is valid in $E\tau$ whenever τ is a specialization of σ and $O\tau$ is true. If, for example, the commutativity axiom $f(x,y) \approx f(y,x)$ is valid in $E\{z/a\}$, this is represented by $f(x,y) \to f(y,x) \ll \langle \{z/a\}, \{x \succ y\} \rangle$. Using these constraints, every equality is orientable.

The completion algorithm has to be provided with a reduction ordering $\succ$ that is complete on the ground terms.[8] The algorithm computes a set $\mathcal{MC}(E, s \approx t)$ of *minimal* rigid E–unifiers. These unifiers are minimal with respect to a (partial) ordering on the substitutions that is induced by $\succ$ (therefore $\mathcal{MC}(E, s \approx t)$ depends on the chosen ordering $\succ$).

Gallier et. al. proved that $\mathcal{MC}(E, s \approx t)$ is always finite (thus its computation always terminates); and that $\mathcal{MC}(E, s \approx t)$ is complete in the following way:

Theorem 3 *If σ is any rigid E–unifier of s and t, then there is a specialization $\bar{\mu}$ of a minimal rigid E–unifier $\mu \in \mathcal{MC}(E, s \approx t)$, such that $\sigma \rightsquigarrow \bar{\mu}$, where the relation $\rightsquigarrow$ on unifiers is defined by*

$$\sigma \rightsquigarrow \bar{\mu} \quad \textit{iff} \quad \textit{for all variables } x \textit{ in } E,s,t\text{: } E\sigma \models \sigma(x) \approx \bar{\mu}(x)$$

In other words, $\sigma \rightsquigarrow \bar{\mu}$ means that $\bar{\mu}$ can be generated from σ using the equations in $E\sigma$. Unfortunately, $\rightsquigarrow$ is not symmetric, as the following example illustrates:

Example 4 *Supposed $E = \{x \approx a\}$, then $\{x/b\} \rightsquigarrow \{x/a\}$, since $\{x \approx a\}\{x/b\} \models b \approx a$, but $\{x/a\} \not\rightsquigarrow \{x/b\}$.*

It is, therefore, not possible to generate a complete set $\mathcal{C}(E, s \approx t)$ of rigid E–unifiers just by applying to minimal unifiers $\mu \in \mathcal{MC}(E, s \approx t)$ equalities from $E\mu$.

Using the set $\mathcal{MC}(E, s \approx t)$, it is, however, possible to efficiently *test* whether a given substitution σ is a rigid E–unifier: σ is a rigid E–unifier of s and t iff there is a $\bar{\mu}$ in the (finite) set $\{\bar{\mu} \ : \ \sigma \rightsquigarrow \bar{\mu}, \bar{\mu} \text{ minimal}\}$ of *minimal* unifiers that can be generated from σ such that there is a $\mu \in \mathcal{MC}(E, s \approx t)$ more general than $\bar{\mu}$.

Based on this test a complete set $\mathcal{C}(E, s \approx t)$ of rigid E–unifiers can be computed in the following way: First, a complete set $\mathcal{P} \supset \mathcal{C}(E, s \approx t)$ of possible rigid E–unifiers is generated by applying the equalities in E in a non–rigid way to the unifiers in $\mathcal{MC}(E, s \approx t)$. All the substitutions in $\mathcal{P}$ are, then, tested; those that are rigid E–unifiers are used to build $\mathcal{C}(E, s \approx t)$.

In practice, and in particular in the semantic tableau framework, it is usually not necessary to generate $\mathcal{P}$, but it suffices to try substitutions that are not taken from $\mathcal{P}$ but that are obtained otherwise, e.g., substitutions that are known to close one or more of the other branches of the tableau.

[7] An order condition is of the form $s \succ t$, where s and t are terms.

[8] $\succ$ is a reduction ordering iff it is well–founded, $s \succ t$ implies $u[s] \succ u[t]$, and $s \succ t$ implies $s\sigma \succ t\sigma$. A reduction ordering that is complete on the ground terms exists for all signatures.

3 Conclusion

To our best knowledge, the method presented here is the first that brings together the advantages of free variable semantic tableaux and those of a completion–based handling of equality. It can easily be combined with recent tableau proving techniques, such as lemma generation or the liberalized δ–rule [7].

A problem that remains to be solved is the combination of rigid and non–rigid E–unification: Given two sets $E_\forall$ and $E_\exists$ of equalities and terms s and t, compute (in a complete way) substitutions σ such that

$$(E_\forall \cup E_\exists\sigma) \models s\sigma \approx t\sigma.$$

A completion–based algorithm solving this problem will allow to efficiently handle equality in tableaux with universal formulas [3].

References

[1] Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. Completion without failure. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2*, chapter 1. Academic Press, 1989.

[2] Bernhard Beckert. Konzeption und Implementierung von Gleichheit für einen tableau–basierten Theorembeweiser. IKBS Report 208, Science Center, Institute for Knowledge Based Systems, IBM Germany, 1991.

[3] Bernhard Beckert and Reiner Hähnle. An improved method for adding equality to free variable semantic tableau. In *Proc. 11th Conference on Automated Deduction CADE, Albany/NY.* Springer, LNCS, 1992.

[4] R. J. Browne. Ground term rewriting in semantic tableaux systems for first–order logic with equality. Technical Report UMIACS–TR–88–44, College Park, MD, 1988.

[5] Melvin C. Fitting. *First–Order Logic and Automated Theorem Proving.* Springer, New York, 1990.

[6] Jean H. Gallier, Paliath Narendran, Stan Raatz, and Wayne Snyder. Theorem proving using equational matings and rigid E–unification. *Journal of the ACM*, 39(2):377–429, April 1992.

[7] Reiner Hähnle and Peter H. Schmitt. The liberalized δ–rule in free variable semantic tableaux. *Journal of Automated Reasoning*, To appear 1993.

[8] R. C. Jeffrey. *Formal Logic: Its Scope and Limits.* McGraw Hill, 1967.

[9] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebras*, pages 263–297. Pergamon Press, Oxford, 1970.

[10] R. J. Popplestone. Beth–tree methods in automatic theorem proving. In *Machine Intelligence*, volume 1, pages 31–46. Oliver and Boyd, 1967.

[11] Steve V. Reeves. Adding equality to semantic tableau. *Journal of Automated Reasoning*, 3:225–246, 1987.

Generating new inference rules for a tableaux-like theorem prover

Catherine Belleannée

IRISA
Campus de Beaulieu 35042 Rennes Cedex
FRANCE
email= belleann@irisa.fr

Abstract

This paper deals with the integration of new connectives (which we call *macro-connectives*) in the framework of tableaux-like theorem provers. Given a new connective, dedicated inference rules (called *macro-rules*) are generated through which the new connective is directly taken into account.

Macro-connectives may be viewed as generic formulas. We consider two kinds of macro-connectives : the ones with *constant arity*, obtained by "variabilizing" formulas, and the ones with *variable arity*, obtained by "generalizing to N" the *homogeneous* formulas (which are strongly regular formulas in some sense). For instance, an homogeneous formula expressing "2-among-3" is to be generalized to N into a macro-connective with variable arity expressing "2-among-N".

We propose a method to induce such macro-connectives from formulas, and then to generate dedicated macro-rules. The method deals with both constant and variable arity cases.

1 Introduction

In a standard tableaux-like propositional theorem provers, the rule system describes the semantics of the primitive connectives: $\wedge, \vee, \Rightarrow, \neg$. Any formula must be expressed in terms of these primitive connectives, so as to be processed by successive application of the appropriate rules. However, for some formulas expressing a "significant" relationship, stating it using primitive connectives may be rather unnatural (cf the "relationships" *equivalent, exclusive-or...*). So, it may be interesting to create new connectives (called *macro-connectives*) dedicated to such relations. Indeed, a macro-connective would be accounted for, in a proof, by a single derivation step through associated inference rule (called *macro-rules*). Moreover, the semantics of the macro-connective would be explicitly rendered by the dedicated macro-rules.

According to this principle, F. Oppacher and E. Suen have integrated some well-known rules, such as *modus ponens*, into their tableaux theorem prover HARP. The rules are called "admissible rules" [Oppacher et al.88].

Among macro-connectives we distinguish the ones with constant arity, and the ones with variable arity. The latter convey some "homogeneity" property, such as "k-among-N". We have developed a method, based on a partial evaluation principle, to generate dedicated inference rules for macro-connectives whether having constant or variable arity.

The basic theorem prover is a tableaux-like cut-free sequent calculus. It is presented in section 2. For a later use, some properties of sequents are presented in section 3.

We then define in section 4 a language for macro-connectives. We also indicate how

macro-connectives with variable arity can be obtained by generalizing to N homogeneous formulas.
Finally, we present a method to generate inference rules dedicated to macro-connectives.

2 Basic formal system

As a basis for developing our approach, we employ a tableaux-like theorem prover with a sequent formalism, namely system G [Gallier86].

Definition : A *sequent* is a pair (Γ, Δ) of, possibly empty, sets of formulas.

Notation: the sequent (Γ, Δ) is usually noted $\Gamma \rightarrow \Delta$, where $\Gamma = \{A1, ..., Am\}$ and $\Delta = \{B1, ..., Bn\}$; the simplified notation is $A1, .., Am \rightarrow B1, .., Bn$.

Semantics : Given an interpretation I, a sequent $A1, .., Am \rightarrow B1, .., Bn$ is *true* in I iff the formula $A1 \wedge ... \wedge Am \Rightarrow B1 \vee ... \vee Bn$ is *true* in I.

2.1 Rules system

As for every tableaux-like theorem prover, the inference rules of the system G directly reflect the semantics of the logical connectives. There are two *inference rules* for each connective: one for formulas occuring in the antecedent of the sequent, and one for formulas occuring in the succedent of the sequent. The rules with only one premise are named α-rules, the others are β-rules.

$$\text{“}\wedge\rightarrow\text{”:}\ \frac{A, B, \Gamma \rightarrow \Delta}{A \wedge B, \Gamma \rightarrow \Delta} \qquad \text{“}\rightarrow\wedge\text{”:}\ \frac{\Gamma \rightarrow \Delta, A \quad \Gamma \rightarrow \Delta, B}{\Gamma \rightarrow \Delta, A \wedge B}$$

$$\text{“}\vee\rightarrow\text{”:}\ \frac{A, \Gamma \rightarrow \Delta \quad B, \Gamma \rightarrow \Delta}{A \vee B, \Gamma \rightarrow \Delta} \qquad \text{“}\rightarrow\vee\text{”:}\ \frac{\Gamma \rightarrow \Delta, A, B}{\Gamma \rightarrow \Delta, A \vee B}$$

$$\text{“}\Rightarrow\rightarrow\text{”:}\ \frac{\Gamma \rightarrow \Delta, A \quad B, \Gamma \rightarrow \Delta}{A \Rightarrow B, \Gamma \rightarrow \Delta} \qquad \text{“}\rightarrow\Rightarrow\text{”:}\ \frac{A, \Gamma \rightarrow \Delta, B}{\Gamma \rightarrow \Delta, A \Rightarrow B}$$

$$\text{“}\neg\rightarrow\text{”:}\ \frac{\Gamma \rightarrow \Delta, A}{\neg A, \Gamma \rightarrow \Delta} \qquad \text{“}\rightarrow\neg\text{”:}\ \frac{A, \Gamma \rightarrow \Delta}{\Gamma \rightarrow \Delta, \neg A}$$

Definition : An *axiom* is any sequent $\Gamma \rightarrow \Delta$ such that Γ and Δ contain some common formula.

Property : Every axiom is valid.

2.2 Deduction tree

Definition : A *deduction tree* for $\Gamma \rightarrow \Delta$ is a tree where :

- the root is labeled with $\Gamma \rightarrow \Delta$
- children nodes are obtained from a parent node using an inference rule. The parent node is labeled with an instance of the conclusion of the rule, and the children nodes are labeled with the corresponding instances of the premises of the rule.

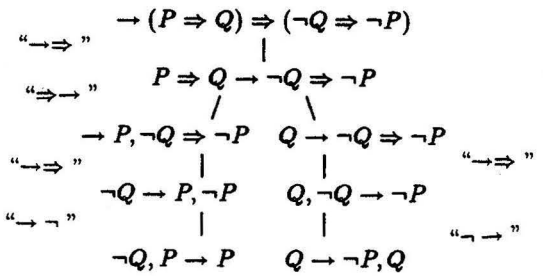


Figure 1: Deduction tree for $\rightarrow (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$

- each *leaf* is labeled with a sequent $\Gamma \rightarrow \Delta$ where:
  - either $\Gamma \rightarrow \Delta$ is an axiom,
  - or Γ and Δ are disjoint sets of atomic formulas.

Notation : "*leaves*$(\Gamma \rightarrow \Delta)$" stands for the set of leaves in the[1] deduction tree for $\Gamma \rightarrow \Delta$.

Definition : A *proof tree* is a finite deduction tree, where all leaves are labeled with axioms.

Theorem : The deduction tree for $(\rightarrow F)$ is a proof tree iff the formula F is valid.

As an example, a deduction tree for the sequent "$\rightarrow (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$" is given in figure 1. This tree is a proof tree establishing the validity of the formula $F = (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$.

3 Some properties and operations on sequents

Here follow some properties and operations on sequents which will be used later on for the construction of the macro-rules.

Definition : a sequent $\Gamma \rightarrow \Delta$ *subsumes* a sequent $\Gamma_1 \rightarrow \Delta_1$ **iff** $\Gamma \subseteq \Gamma_1$ and $\Delta \subseteq \Delta_1$.

Definition : *rule of transitivity on sequents*
Given $\Gamma, \Delta, \Gamma_1, \Delta_1$ four finite sets of formulas and F a formula,
$(\Gamma \rightarrow F, \Delta)$ and $(F, \Gamma_1 \rightarrow \Delta_1)$ *imply* $(\Gamma, \Gamma_1 \rightarrow \Delta, \Delta_1)$ *by transitivity* .

Definition : S *follows from* E *by transitivity* iff there exist $S_1, ..., S_n$ such that

- $S_1 \in E$
- for $i = 1...n-1$, there exists Seq in $E \cup \{S_j, j \leq i\}$ such that S_i and Seq imply S_{i+1} by transitivity
- $S_n \equiv$ S

Definition : operator $+_s$: *sequent fusion*
The fusion of two sequents $\Gamma \rightarrow \Delta$ and $\Gamma' \rightarrow \Delta'$ gives the sequent $\Gamma, \Gamma' \rightarrow \Delta, \Delta'$
example: $a, b \rightarrow c \ +_s \ g \rightarrow h, i \ = \ a, b, g \rightarrow c, h, i$

Definition : operator $\times_s$: *product-fusion*
Given $\mathcal{S}_1$ and $\mathcal{S}_2$ two sets of sequents the *product-fusion* of $\mathcal{S}_1$ and $\mathcal{S}_2$ produces the set of sequents: $\mathcal{S}_1 \times_s \mathcal{S}_2 = \{ S_1 +_s S_2 \ / \ (S_1, S_2) \in \mathcal{S}_1 \times \mathcal{S}_2 \}$.

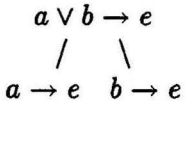
[1] We speak of *the* deduction tree for a sequent because we assume a fixed strategy

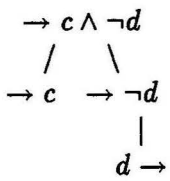
Example : Given $\mathcal{S}_1 = \{(a, b \rightarrow c),\ (d \rightarrow e)\}$ and $\mathcal{S}_2 = \{(f \rightarrow),\ (g \rightarrow h, i)\}$,
$\mathcal{S}_1 \times_s \mathcal{S}_2 = \{(a, b, f \rightarrow c),\ (a, b, g \rightarrow c, h, i),\ (d, f \rightarrow e),\ (d, g \rightarrow e, h, i)\}$.

Proposition :
If $Seq = Seq_1 +_s Seq_2$ then $leaves(Seq) \equiv leaves(Seq_1) \times_s leaves(Seq_2)$,
where $\equiv$ stands for "equality of sets modulo subsumptions" .

Macro-rules for a macro-connective M are obtained from the sets of leaves in the deduction trees for $(M \rightarrow)$ and $(\rightarrow M)$. Thus, the above proposition permits to generate the rules for M from the rules for the sub-formulas of M. We give more details later on.
A similar result about tableaux is used by E. Lafon and C. Schwind [Lafon et al.88] to verify incrementally the consistency of a set of formulas.

Example : Let S = $(a \vee b \rightarrow c \wedge \neg d, e)$. We have S = $(a \vee b \rightarrow e) +_s (\rightarrow c \wedge \neg d)$. According to the previous proposition, *leaves*(S) can be obtained from *leaves*$(a \vee b \rightarrow e)$ and *leaves*$(\rightarrow c \wedge \neg d)$ (i.e. without explicitly developing the deduction tree for $(a \vee b \rightarrow c \wedge \neg d, e)$):





That is, *leaves*$(a \vee b \rightarrow e) = \{a \rightarrow e,\ b \rightarrow e\}$ and *leaves*$(\rightarrow c \wedge \neg d) = \{\rightarrow c,\ d \rightarrow\}$
So, *leaves*(S) $= \{a \rightarrow e,\ b \rightarrow e\} \times_s \{\rightarrow c,\ d \rightarrow\}$
$= \{a \rightarrow e, c,\ a, d \rightarrow e,\ b \rightarrow e, c,\ b, d \rightarrow e\}$

4 Macro-connectives

Macro-connectives may be viewed as generic formulas. We introduce the notion of a "partition" to define formally the concept of generic formulas. Intuitively, the partitions of a formula correspond to the various levels of abstraction of the formula.

Definition : A *partition* of a formula F *with respect to some arguments FL* is a λ-term (A FL) such that
- A is a λ-abstraction without free variables,
- FL is a list of formulas,
- the β-reduction of (A FL) gives F.

A is called an *abstraction* of F with respect to the arguments FL.

For example, $((\lambda XYZ.(X \Rightarrow Y) \Rightarrow Z))\ \ p\,q\,r$, $((\lambda XY.X \Rightarrow Y))\ (p \Rightarrow q)\ r$ and $((\lambda X.X))\ (p \Rightarrow q) \Rightarrow r$ are the different partitions for the formula $(p \Rightarrow q) \Rightarrow r$.

4.1 Macro-connectives with constant arity

Definition : A *macro-connective with constant arity k* is an abstraction of a formula with respect to k arguments.

Examples :

- $\lambda X.X$ is the identity macro-connective, with arity 1.

- $\lambda XY.(X \Rightarrow Y) \wedge (Y \Rightarrow X)$ is a macro-connective with arity 2 corresponding to the usual equivalence relation.
- $\lambda XYZ.(X \Rightarrow Y) \wedge (X \Rightarrow Z)$ is a macro-connective with arity 3.

4.2 Macro-connectives with variable arity

Generic macro-connectives, namely *macro-connectives with variable arity*, are obtained by generalizing the number of parameters in a macro-connective. Expressing such objects requires the definition of a language of connectives dealing with *sets* of formulas.
The simplest ones are *primitive macro-connectives with variable arity.* Given a commutative and associative connective C, such a macro-connective results from generalizing C to the case of an arbitrary number of arguments. The extension of C is noted $\overset{\infty}{C}$.
So, $\overset{\infty}{\wedge}(\{ a, b, c\})$ stands for the application of $\wedge$ to a, b, c in any order.
More generally, a macro-connective with variable arity may have various instances of a macro-connective (see the definition of *pattern*) as a set of arguments.
A new symbol, $\overset{\infty}{\subset}$, is used to describe those generic sets of instances for a pattern.
$\overset{\infty}{\subset}$*(Pattern, Set-of-formulas)* stands for the set of all the instances for the macro-connective *Pattern*, taking their value in *Set-of-formulas*, without redundancies (i.e. only one instance among equivalent ones modulo permutation is generated). For more details, see [Belleannée91].
For example:
at-least-two-false $(N) = \overset{\infty}{\vee}\ (\overset{\infty}{\subset}\ (\lambda XY.\neg X \wedge \neg Y, N))$ is a macro-connective with variable arity.
at-least-two-false $(\{p, q\}) = (\neg p \wedge \neg q)$
at-least-two-false $(\{p, q, r\}) = (\neg p \wedge \neg q) \vee (\neg p \wedge \neg r) \vee (\neg q \wedge \neg r)$.

4.3 Homogeneity

Homogeneity is a property of strong syntactical regularity of the subformulas of a formula. Generalizing this regularity to any number of subformulas will produce a generic formula with variable arity, i.e. a macro-connective with variable arity.

Definition : A formula is an *homogeneous block with respect to a set of components* $\mathcal{E}$, if there exists a partition $(A\ (M\ LF_1)...(M\ LF_n))$ of F such that
- $\mathcal{E} = \cup_i\ LF_i$
- A has the form $\lambda X_1 \cdots X_n.\ X_1\ C \cdots C\ X_n$, with C a commutative and associative connective,
- the set $\{(M\ LF_i)\}$ is syntactically invariant for every permutation of the components $\mathcal{E}$.

M is the *pattern* and C the *link* of the block . F is denoted $(< C, M >, \mathcal{E})$.

Definition : F is an *homogeneous formula* if there exists a partition $(A\ F_1...F_n)$ of F such that each F_i is an homogeneous block with respect to a same set of components $\mathcal{E}$. Homogeneous blocks are the basic homogeneous formulas.

Examples :

- $F_0 = (a \wedge b) \Rightarrow ((c \wedge d) \Rightarrow (e \wedge c))$ is not a homogeneous block because $\Rightarrow$ is not commutative.
- $F_1 = p \vee q \vee r$ is a homogeneous block with respect to $\{p, q, r\}$, denoted $(< \vee, Id >, \{p, q, r\})$.

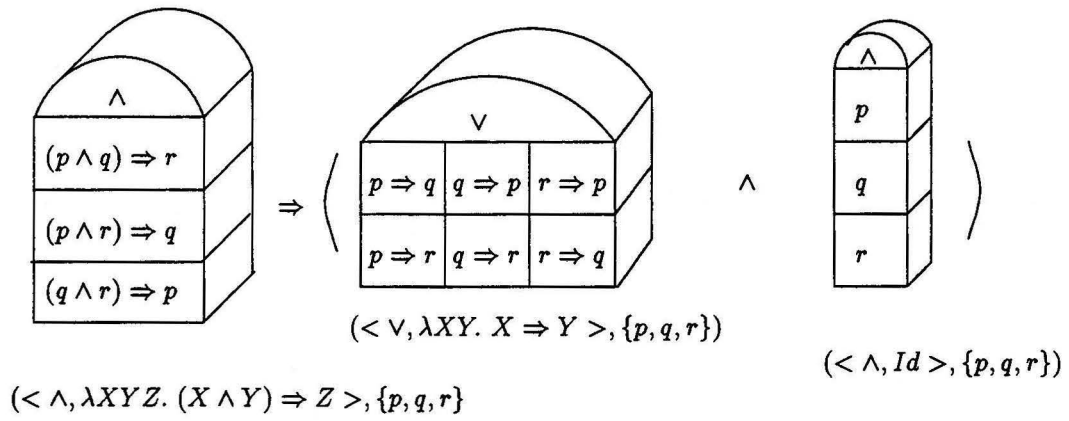


Figure 2: Block structure for F_5.

- $F_2 = (\neg p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg q \vee \neg r)$ is a homogeneous block with respect to $\{p, q, r\}$, denoted $(< \wedge, \lambda XY.\neg X \vee \neg Y >, \{p, q, r\})$.
- $F_3 = (\neg p \vee \neg q) \wedge (\neg q \vee \neg r)$ is not a homogeneous block with respect to $\{p, q, r\}$. Indeed, look at the permutation $\sigma = \{p \leftarrow p, r \leftarrow q, q \leftarrow r\}$. Clearly, $\sigma F_3 = (\neg p \vee \neg r) \wedge (\neg r \vee \neg q)$ is not equivalent to F_3.
- $F = (p \vee q \vee r) \wedge (\neg p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg q \vee \neg r)$ is not a homogeneous *block* with respect to $\{p, q, r\}$, but it is a homogeneous *formula* with respect to $\{p, q, r\}$. Its block structure is $F = F_1 \wedge F_2$. F will be denoted $(< \vee, Id >, \{p, q, r\})$ $\wedge$ $(< \wedge, \lambda XY.\neg X \vee \neg Y >, \{p, q, r\})$
- $F_5 = (((p \wedge q) \Rightarrow r) \wedge ((p \wedge r) \Rightarrow q) \wedge ((q \wedge r) \Rightarrow p)) \Rightarrow (((p \Rightarrow q) \vee (p \Rightarrow r) \vee (q \Rightarrow p) \vee (q \Rightarrow r) \vee (r \Rightarrow p) \vee (r \Rightarrow q)) \wedge p \wedge q \wedge r)$ is a homogeneous *formula* with respect to $\{p, q, r\}$. Its block structure is depicted in figure 2.

4.4 Homogeneous macro-connectives with variable arity

Homogeneity is based on a regular stucture (block structure, with patterns) which can be naturally extended to deal with an arbitrary number of arguments. This is done by "generalizing to N" [2] the number of arguments of a homogeneous formula, while preserving the block structure. This results in a homogeneous macro-connective with variable arity. We now define this mapping.

Definition : Given F, a homogeneous formula with respect to the set $\mathcal{E}$, and given $(A.\ F_1 ... F_k)$ a partition of F, where every F_i is a homogeneous block, the *macro-connective* $\overset{\infty}{F}(N)$*, with variable arity induced from* F, is the partition $(A.\ \overset{\infty}{F}_1\ (\mathrm{N}) ...\ \overset{\infty}{F}_k(\mathrm{N}))$, where each $\overset{\infty}{F}_i\ (N)$ is defined from the block F_i in the following way:
if F_i is a block denoted $(< C_i, M_i >, \mathcal{E})$ then $\overset{\infty}{F}_i\ (N) = \overset{\infty}{C}_i\ (\overset{\infty}{=}\ (M_i, N))$

So, $\overset{\infty}{F}_i\ (N)$ is an extension of the block F_i preserving the block structure $< C_i, M_i >$ of F_i. That is, every instance $\overset{\infty}{F}_i\ (\mathcal{E})$ of $\overset{\infty}{F}_i\ (\mathrm{N})$ is a homogeneous block with respect to $\mathcal{E}$, denoted $(< C_i, M_i >, \mathcal{E})$. So, $\overset{\infty}{F}_i\ (N)$ is a *generic* homogeneous block taking a set as a parameter. It will be noted $< C_i, M_i >$.

[2] Generalization to N is an induction method. It consists in introducing a loop at places where a repetition is detected [Cohen88, Shavlick90]

Example : Let the formula $F = (p \vee q \vee r) \wedge (\neg p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg q \vee \neg r)$ homogeneous with respect to $\mathcal{E} = \{p, q, r\}$, with block structure $F = F_1 \wedge F_2$ where $F_1 = (< \vee, Id >, \{p, q, r\})$ and $F_2 = (< \wedge, \lambda XY.\neg X \vee \neg Y >, \{p, q, r\})$, its associated macro-connective $\overset{\infty}{F}(N)$ with variable arity is $\overset{\infty}{F}(N) = \overset{\infty}{F_1}(N) \wedge \overset{\infty}{F_2}(N)$ where

- $\overset{\infty}{F_1}(N) = < \vee, Id >$ that is $\overset{\infty}{F_1}(N) = \overset{\infty}{\vee}(\overset{\infty}{\subset}(Id, N)) = \overset{\infty}{\vee}(N)$.
- $\overset{\infty}{F_2}(N) = < \wedge, \lambda XY.\neg X \vee \neg Y >$ that is $\overset{\infty}{F_2}(N) = \overset{\infty}{\wedge}(\overset{\infty}{\subset}(\lambda XY.\neg X \vee \neg Y, N))$

and finally $\overset{\infty}{F}(N) = \overset{\infty}{\vee}(N) \wedge \overset{\infty}{\wedge}(\overset{\infty}{\subset}(\lambda XY.\neg X \vee \neg Y, N))$.

For example, $\overset{\infty}{F}(\{a, b, c, d\}) = (a \vee b \vee c \vee d) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg a \vee \neg d) \wedge (\neg b \vee \neg c) \wedge (\neg b \vee \neg d) \wedge (\neg c \vee \neg d)$.

5 Dedicated rules

This section deals with the integration of the new connectives in the framework of the system G. Given a new connective, dedicated inference rules (called *macro-rules*) are generated through which the new connective is directly taken into account.
Similarly to primitive connectives, each macro-connective will give rise to two inference macro-rules. One deals with an occurrence of the macro-connective in the left part of the sequent, and the other deals with an occurrence of the macro-connective in the right part of the sequent.
A macro-rule is produced by synthesizing the deduction trees associated to he macro-connective. The method used to generate the inference rules assigned to a macro-connective "MC" may be described by the following algorithm:

```
func construct-the-rules (MC) :
       construct-one-rule ( -> MC)
       construct-one-rule ( MC -> )

func construct-one-rule (Sequent) :
       deduction-tree <- expand-tree(Sequent)
       premise <-  reduce (leaves (deduction-tree))
                   premise
       RESULT <-  --------
                   Sequent

func reduce (leaves)
       int-leaves  <- suppress-axioms (leaves)
       RESULT      <- suppress-redundancies (int-leaves)
```

5.1 Generating new inference rules: constant arity case

For a macro-connective MC with constant arity, the function "expand-tree" produces the deduction tree for the sequent $(MC \rightarrow)$ (then $(\rightarrow MC)$). The deduction tree is then pruned by discarding all axiom leaves and *redundant* leaves. The resulting macro-rule consists of the leaves of the pruned tree as its premises, and the macro-connective structure as its conclusion.
For example, given the macro $MC = \lambda XYZ\ (X \Rightarrow Y) \wedge (X \Rightarrow Z)$ the different steps in constructing "$MC \rightarrow$" (resp. "$\rightarrow MC$") are given in figure 3 (resp. figure 4).

5.1.1 Redundancies

Elimination of redundancies in the set of leaves is a critical matter in the construction of the rules. In this respect, the main result characterizes the implication of sequents (a semantical notion) in terms of subsumption and transitivity (i.e. a syntactical notion).

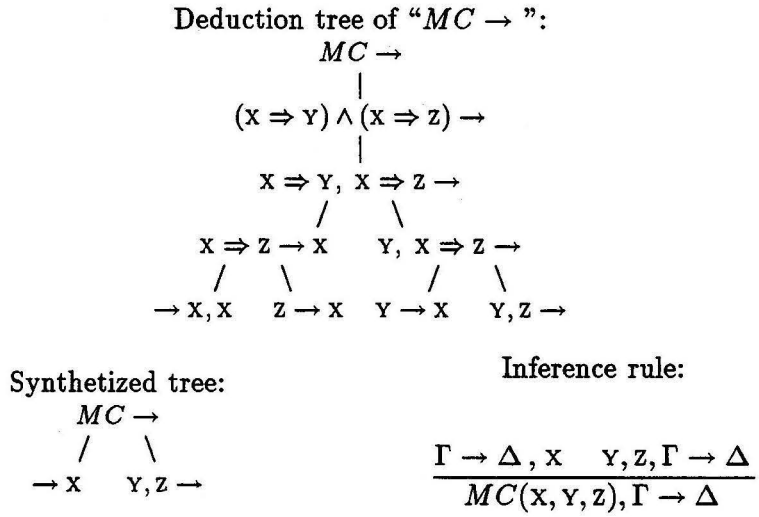
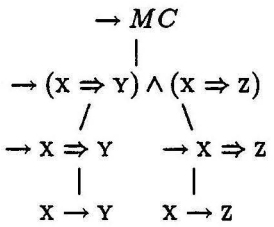
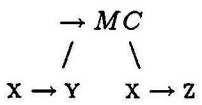


Figure 3: Construction of "$MC \rightarrow$"

Deduction tree of "$\rightarrow MC$":



Synthetized tree:



Inference rule:

$$\frac{X, \Gamma \rightarrow \Delta, Y \quad X, \Gamma \rightarrow \Delta, Z}{\Gamma \rightarrow \Delta, MC(X, Y, Z)}$$

Figure 4: Construction of "$\rightarrow MC$"

Definition : A sequent S is *redundant* with respect to a set E of sequents **iff** S is implied by E. That is, taking $E = S_1, \cdots, S_n$, S is redundant with respect to E iff for all I, if $I \models \bigwedge_i S_i$ then $I \models S$.

Proposition : A sequent S is *redundant with respect to a set $\mathcal{E}$ of leaf sequents* **iff** there exists a sequent S_n such that S_n *follows* from $\mathcal{E}$ *by transitivity*, and S_n *subsumes* S. The proof is given in [Belleannée91].

Example : The set of leaves $\{a \rightarrow a, b, \quad b \rightarrow c, d, \quad c, a \rightarrow d, \quad a, b \rightarrow d, e\}$ is reduced into $\{b \rightarrow c, d, \quad c, a \rightarrow d\}$, after eliminating the axiom $a \rightarrow a, b$ and the redundant sequent $a, b \rightarrow d, e$ (implied by transitivity by $b \rightarrow c, d$ and $c, a \rightarrow d$) :
$b \rightarrow c, d, \quad c, a \rightarrow d \xmapsto{T} a, b \rightarrow d \xmapsto{S} a, b \rightarrow d, e.$

5.2 Generating new inference rules: variable arity case

In order to derive the deduction tree for a macro-connective with variable arity according to the system G, the function "expand-tree" needs extended derivation rules supporting variable arity. We do not give here all the details of the method, but only the mains steps of the algorithm. For illustration purpose, we give the result for the generic block $\overset{\infty}{F}_2 = < \wedge,\ \lambda XY.\ \neg X \vee \neg Y >$.

In the following, the premise of a rule R is denoted $\mathcal{P}_R$.

1. Deduction of the pattern M:
   generation of the rules "$M \rightarrow$" and "$\rightarrow M$", using the previous algorithm.
   In the example, M = (λXY. $\neg$X $\vee$ $\neg$Y)

   Deduction of ($\neg$X $\vee$ $\neg$Y $\rightarrow$):

   $\neg X \vee \neg Y \rightarrow$

   / \

   $\neg X \rightarrow$ $\quad$ $\neg Y \rightarrow$

   | $\quad$ |

   $\rightarrow X$ $\quad$ $\rightarrow Y$

   That is $\mathcal{P}_{M\rightarrow} = \{\rightarrow X\ ,\ \rightarrow Y\}$

   Deduction of ($\rightarrow \neg$X $\vee$ $\neg$Y):

   $\rightarrow \neg X \vee \neg Y$

   |

   $\rightarrow \neg X, \neg Y$

   |

   $X, Y \rightarrow$

   That is $\mathcal{P}_{\rightarrow M} = \{X, Y \rightarrow\}$

2. Deduction of the block $\overset{\infty}{F}_i = < C, M >$.

   Rule "$\overset{\infty}{F}_i\rightarrow$":
   - **if** the type of the rule "$C \rightarrow$" is β **then** $\mathcal{P}_{\overset{\infty}{F}_i\rightarrow} \equiv \overset{\infty}{\bigcirc}(\mathcal{P}_{M\rightarrow}, \mathrm{N})$,
   - **if** the type of the rule "$C \rightarrow$" is α **then** $\mathcal{P}_{\overset{\infty}{F}_i\rightarrow} \equiv \mathrm{PROD}^3\ (\mathcal{P}_{M\rightarrow}, \mathrm{N})$.

   Rule "$\rightarrow\overset{\infty}{F}_i$" : following the same principle.

   Here is the critical step of the algorithm: we now have to deal with sets with variable size. The main result is given by the following proposition, which indicates how to compute product-fusion involving a variable number of sequents.

   Proposition: if $\mathcal{P}$ ($= \mathcal{P}_{\rightarrow M}$ or $\mathcal{P}_{M\rightarrow}$) is a set of sequents, of which a sequents have an empty right part, b sequents have an empty left part, and there are c remaining sequents,
   then $\mathrm{PROD}(\mathcal{P}, \mathrm{N}) \equiv$
   $$\overset{\infty}{\bigcirc}((N - [X_1, ..., X_{a-1}] \rightarrow), \mathrm{N}) \quad \bigcup \quad \overset{\infty}{\bigcirc}((\rightarrow N - [X_1, ..., X_{b-1}]), \mathrm{N})$$

   In the example, $\overset{\infty}{F}_2 = < \wedge,\ \lambda XY.\neg X \vee \neg Y >$
   the type of "$\wedge \rightarrow$" is α, so
   $\mathcal{P}_{\overset{\infty}{F}_2\rightarrow} \equiv \mathrm{PROD}(\mathcal{P}_{M\rightarrow}, \mathrm{N})$
   $\equiv \mathrm{PROD}(\{ \rightarrow X\ ,\ \rightarrow Y\ \}, \mathrm{N})$
   $\equiv \overset{\infty}{\bigcirc}((\rightarrow N - [X]), \mathrm{N})$.

   the type of "$\rightarrow \wedge$" is β, so
   $\mathcal{P}_{\rightarrow\overset{\infty}{F}_2} \equiv \overset{\infty}{\bigcirc}\ (\mathcal{P}_{\rightarrow M}, \mathrm{N})$
   $\equiv \overset{\infty}{\bigcirc}\ ((X, Y \rightarrow), \mathrm{N})$.

[3] "PROD" stands for the extension to a set of arguments of the binary operator "product-fusion"

So, given N = {a,b,c,d}, by instanciating $\mathcal{P}_{\rightarrow \overset{\infty}{F_2}}$ and $\mathcal{P}_{\overset{\infty}{F_2}\rightarrow}$ we obtain:

$\mathcal{P}_{\rightarrow \overset{\infty}{F_2}} \equiv \{(a,b\rightarrow),\ (a,c\rightarrow),\ (a,d\rightarrow),\ (b,c\rightarrow),\ (b,d\rightarrow),\ (c,d\rightarrow)\}$

$\mathcal{P}_{\overset{\infty}{F_2}\rightarrow} \equiv \{(\rightarrow a,b,c),\ (\rightarrow a,b,d),(\rightarrow a,c,d),\ (\rightarrow b,c,d)\}$

A standard deduction for the sequent $\overset{\infty}{F_2}\rightarrow$ would have produced a tree with 516 nodes and a depth of 17.

5.2.1 Rules dedicated to a homogeneous formula with variable arity

Given $\overset{\infty}{F}$, a homogeneous formula such that $\overset{\infty}{F} = (\mathrm{A}\ \overset{\infty}{F_1} \ldots\ \overset{\infty}{F_n})$, where each $\overset{\infty}{F_i}$ is an homogeneous block, the computation of the macro-rules dedicated to $\overset{\infty}{F}$ follows two steps:
- first the deduction of each $\overset{\infty}{F_i}$ is done, according to the previous algorithm, producing the sets $\mathcal{P}_{\overset{\infty}{F_i}\rightarrow}$ and $\mathcal{P}_{\rightarrow \overset{\infty}{F_i}}$
- then the resulting sets of leaves are merged to constitute $\overset{\infty}{F}\rightarrow$ and $\rightarrow\overset{\infty}{F}$.

We do not give the details of the merging. We only show the result of the computation on an example.

Example: Let $\overset{\infty}{F} = <\vee, Id> \wedge <\wedge, \lambda \mathrm{XY}.\ \neg\mathrm{X} \vee \neg\mathrm{Y}>$.

$\overset{\infty}{F}$ represents the concept "exclusive or with arity N".

$((\lambda \mathrm{X_1X_2}.\ \mathrm{X_1} \wedge \mathrm{X_2})\ \overset{\infty}{F_1}\overset{\infty}{F_2})$ is the block partition of $\overset{\infty}{F}$. $\overset{\infty}{F_1}$ and $\overset{\infty}{F_2}$ are generic blocks noted $<\vee, Id>$ and $<\wedge, \lambda \mathrm{XY}.\ \neg\mathrm{X} \vee \neg\mathrm{Y}>$, respectively .

1- The macro-rules dedicated to $\overset{\infty}{F_1}$ and $\overset{\infty}{F_2}$ obtained by means of the previous algorithm are such that:

$\mathcal{P}_{\overset{\infty}{F_1}\rightarrow} = \overset{\infty}{\sim}((\mathrm{X}\rightarrow), \mathrm{N})$ and $\mathcal{P}_{\rightarrow \overset{\infty}{F_1}} = \{(\rightarrow N)\}$

$\mathcal{P}_{\overset{\infty}{F_2}\rightarrow} = \overset{\infty}{\sim}((\rightarrow N - [\mathrm{X}]), \mathrm{N})$ and $\mathcal{P}_{\rightarrow \overset{\infty}{F_2}} = \overset{\infty}{\sim}((\mathrm{X,Y}\rightarrow), \mathrm{N})$

2- The merging to constitute the macro-rules dedicated to $\overset{\infty}{F}$ gives:

Rule "$\overset{\infty}{F}\rightarrow$"

$$\begin{aligned}\mathcal{P}_{\overset{\infty}{F}\rightarrow} &\equiv \mathcal{P}_{\overset{\infty}{F_1}\rightarrow} \times_s \mathcal{P}_{\overset{\infty}{F_2}\rightarrow} \\ &\equiv \overset{\infty}{\sim}((\mathrm{X}\rightarrow), \mathrm{N}) \times_s \overset{\infty}{\sim}((\rightarrow N - [\mathrm{X}]), \mathrm{N}). \\ &\equiv \overset{\infty}{\sim}((\mathrm{X} \rightarrow N - [\mathrm{X}]), \mathrm{N}).\end{aligned}$$

This establishes the generic rule "$\overset{\infty}{F}\rightarrow$" as

$$\frac{\overset{\infty}{\sim}((\mathrm{X} \rightarrow N - [\mathrm{X}]), N)}{\overset{\infty}{F}\rightarrow}$$

that is, "$\overset{\infty}{F}$ is *true* if one among the elements of N is *true* and all the others are *false*".

Rule "$\rightarrow\overset{\infty}{F}$"

$$\begin{aligned}\mathcal{P}_{\rightarrow\overset{\infty}{F}} &\equiv \mathcal{P}_{\rightarrow\overset{\infty}{F_1}} \cup \mathcal{P}_{\rightarrow\overset{\infty}{F_2}} \\ &\equiv \{(\rightarrow N)\} \cup \overset{\infty}{\sim}((\mathrm{X,Y}\rightarrow), \mathrm{N}).\end{aligned}$$

This establishes the generic rule "$\rightarrow\overset{\infty}{F}$" as

$$\frac{\{(\rightarrow N)\} \cup \overset{\infty}{\sim}((\mathrm{X,Y}\rightarrow), N)}{\rightarrow\overset{\infty}{F}}$$

that is, " $\overset{\infty}{F}$ is *false* if all the elements of N are *false* or at least two are *true*".

Let N= {a,b,c}. Then,

$$\mathcal{P}^{\infty}_{F\rightarrow} \equiv \{(a \rightarrow b, c),\ \ (b \rightarrow a, c),\ \ (c \rightarrow a, b)\}$$

$$\mathcal{P}^{\infty}_{\rightarrow F} \equiv \{(\rightarrow a, b, c),\ \ (a, b \rightarrow),\ \ (a, c \rightarrow),\ \ (b, c \rightarrow)\}$$

6 Conclusion

The method presented above allows the user of the theorem prover to introduce his own macro-connectives, for which specific rules are generated.
In our opinion, an interesting point of the macro-rule notion is to propose a means to generate automatically new rules, when they appear useful in some application domain. In constrast, it seems foolish to work with a "universal" system of rules obtained from adding a quasi-exhaustive collection of new inferences rules. Indeed, such a system would be frozen, as the standard one, and there surely would exist some domain exhibiting a useful, yet "non universal", relationship not captured by that system. So, the idea is to set up a dedicated system for each application domain. The construction of macro-rules is not costly (rules are computed once for all) and the use of these rules can greatly decrease the depth of proof trees (by saving time and space in deductions).
A broader perspective to this work is to produce automatically a set of macro-rules for a specific domain, in view of a sample of theorems in the domain. The ideal result would be to generate, for any domain, a system of rules with all and only useful rules (to lower the amount of time wasted in searching for a rule to apply). An underlying critical point is to develop criteria to decide what a useful rule is.

References

[Belleannée91] Belleannée (C.). – *Vers un démonstrateur de théorèmes adaptatif.* – Thèse de l'université de Rennes 1, janvier 1991.

[Cohen88] Cohen (W.W.). – Generalizing Number and Learning from Multiple Examples in Explanation Based Learning. *In: 5th ICML*, éd. par Kaufmann (Morgan), pp. 256–269.

[Gallier86] Gallier (J.H.). – *Logic for Computer Science: Foundations of Automatic Theorem Proving.* – New York, Harper and Row, 1986.

[Lafon et al.88] Lafon (E.) and Schwind (C.). – A Theorem Prover for Action Performance. *In: proceedings of ECAI*, éd. par Pitman.

[Oppacher et al.88] Oppacher (F.) and Suen (E.). – HARP:A Tableau-based Theorem Prover. *Journal of Automated Reasoning*, vol. 4, 1988, pp. 69–100.

[Shavlick90] Shavlick (J.W.). – Acquiring Recursive and Iterative Concepts with Explanation-Based Learning. *Journal of Machine Learning*, vol. 5, 1990, pp. 39–70.

A Three-Dimensional Refinement for Tableaux

Krysia Broda[1]

1. Introduction Consider the following problem: $\Delta \models \exists x P[x]$. We want

(1) To show that indeed the above holds using a proof procedure
(2) To uniformly generate schematic solutions for witnesses or disjunctive substitutions which make it true.

The second requirement is dependent on the proof procedure and some procedures are more amenable than others. The traditional tableaux systems are not very friendly for (2) for they require a search of every possible tableau to be sure that all solutions have been found. The purpose of this note is to present a three dimensional refinement of tableaux which more naturally yields (2).

It is traditional in tableau systems to use heuristics to derive a tableau refutation that is as concise as possible. However, such a search yields only one 'solution' to the original problem and we are looking for all of them. Thus it would seem as though many tableaux must be formed even though it is likely that amongst the refutations there are many that are similar. Two tableaux are similar if they use the same set of universal-type (constituents of) sentences but with different substitutions, or, if they use exactly the same sentences and substitutions but taken in a different order. Ideally, one would like the generation of similar tableaux to be avoided as far as possible. Further, there are likely to be many common parts of the various tableaux that are formed, and it would be beneficial if these can be shared as well. The method addressed in this note is aimed at achieving this.

There are two approaches towards achieving our aim:

(a) Add unification to tableaux: this helps to solve the first difficulty and often results in a schematic witness being found. For example, $\{Q(a), \forall y Q(f(y)), \forall x[Q(x) \rightarrow P(x)]\} \models P(a), P(f(a)), P(f(f(a)))$, etc. and using unification the schematic witness f(y1) would be obtained for the conclusion $\exists x P(x)$.

(b) Restrict the tableaux that can be formed: the selection of possible steps in a tableau is controlled, so that fewer tableaux need to be investigated to find all solutions. For example a linear restriction could be imposed, in which at each step one of the sentences added at the previous step must be expanded and at least one branch must be made to close. For the sentences above this would result in $\forall x[Q(x) \rightarrow P(x)]$ being used below the negated conclusion $\forall x \neg P(x)$. A consequence of the imposition of a refinement is that some attempts at forming a closed tableau may end in failure, there being no more steps of the allowed sort that can be applied to a branch. This means some backtracking is required in order to try an alternative. A further problem with refinements is that they may still result in several tableaux being formed in the search space that use the same sentences and substitutions for universal variables.

In this note a flexible semi-linear refinement is described that makes use of three dimensional tableaux. This refinement has an additional advantage in that there is a book-keeping method which can be imposed upon it to so as to restrict the formation of similar tableaux. The book-keeping is an adaptation of the method of connection graphs [Kowalski] which was first introduced for clausal deduction. The method is restricted to clausal tableaux here, but by introducing a compilation phase it can be extended to general sentences.[See Broda].

The main features of the new refinement are:

(i) Unification is incorporated into the tableau rule for universal sentences

---

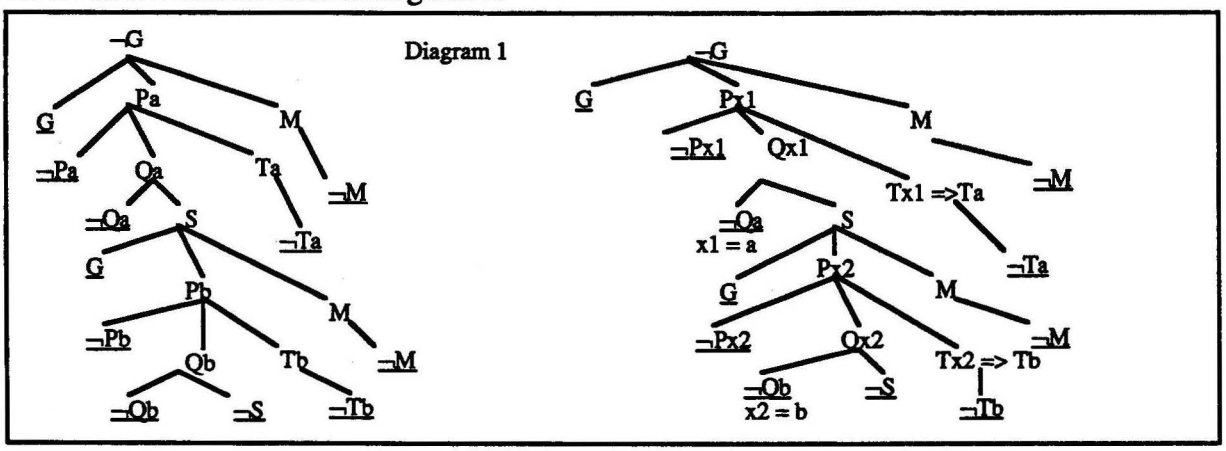
[1]Dept of Computing, Imperial College,180 Queens Gate, London SW7 2BZ email: kb@doc.ic.ac.uk

(ii) The semi-linear refinement is an extension of the linear refinement
(iii) A node in a tableau may be extended in more than one way, giving rise to a third dimension
(iv) Common sub-tableaux are implicitly shared

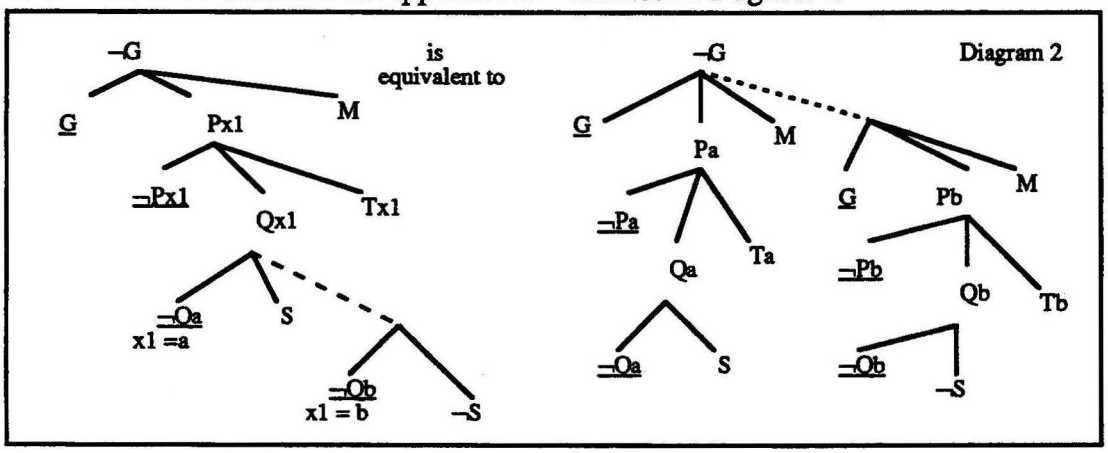
For ***complete*** refinements, every tableau using a different set of substitutions (and sentences from the Data) will be formable within the rules of the refinement and in one search space. The semi-linear refinement is complete in this sense.

Unification in tableaux affects the expansion of a universal quantified sentence and branch closure. A new expansion rule for a universal sentence ∀xP[x] yields P[x1], where all occurrences of x in P[x] are replaced by x1, called a dummy variable. The dummy variable represents an unknown term, and it becomes bound to a particular term when a suitable one is found by branch closure. A branch may be closed if two literals in the branch can be made complementary by unifying the literals. The most general unifier, which generally will include bindings for some dummy variables, is then applied to occurrences of these dummy variables throughout the tableau.

We illustrate the method for a simple example, see Diagrams 1 and 2. A closed traditional tableau for the clauses {¬G, GPxM, ¬PxQxTx, ¬QaS, ¬Qb¬S,¬Ta,¬Tb,¬M} is shown as the first tableau in diagram 1.



¬QaS uses a linear strategy: each leaf literal is matched with a literal in the next clause to be used. The three-dimensional approach is illustrated in diagram 2.



The right hand tableau is the ground tableau in which the dotted line indicates an extension into a third dimension. The two sub-tableaux are called planes and each is linear and has the same structure. The closure below M will be shared. The left hand tableau indicates the dynamic formation of the tableau. Below Qx1 there are two possible extension steps, one

using ¬QaS and the other using ¬Qb¬S. These give rise to two planes, the substitution for x1 in the first being a, and in the second being b, which share the development up to this point. They can then be combined to derive the right hand tableau in diagram 1.

2. The Semi-linear Refinement for Clauses and Three–Dimensional Tableaux

A: A clausal tableau is a tree structure of nodes each labelled by a literal and formed by using the clausal tableau rule and closure rule. For uniformity, the first clause used in the tableau, called the top clause, has a literal G not occurring in any other clause disjoined to it and the root node is labelled by ¬G. All input clauses are assumed to be available in the tableau. Each node is is related to its immediate descendants by the tree relation <. The transitive closure of <, <*, relates nodes in a branch and a subtree: a branch with leaf node k={nodes n: n ≤*k } and the subtree rooted at n={nodes m: m≥*n}.

The clausal tableau rule develops a clause $\forall \underline{x} P1(\underline{x}) \vee \ldots \vee Pn(\underline{x})$ below a leaf node k by extending the tree at k by n branches and n new nodes labelled by $Pi(\underline{x1})$, where $\underline{x1}$ is a vector of new dummy variables. k is < each new node. The closure rule allows a branch to be closed if its leaf literal unifies with another node in the branch. The substitutions for the dummy variables are propagated throughout the tableau.

B: A three-dimensional clausal tableau is a clausal tableau in which each node n is also labelled by a set of constants P(n), naming the planes it is in. For an ordinary clausal tableau, all nodes are in the same, single, plane and so there is no need for the additional label. The constants P are a subset of Planes, a set of constants disjoint from any constants used in the language from which the input sentences are formed. P(n) will usually increase as the tableau is formed. If P(k) =C, then for all nodes m belonging to the subtree rooted at k, P(m) ⊆C and for each c ∈ C, all nodes n: c ∈ P(n), P(n)⊇C.

For fixed c ∈ C, the plane c={nodes k: c ∈ P(k)} and is an ordinary clausal tableau. Each plane c has an associated substitution for dummy variables. The ancestry (k) in plane p = {nodes m: m is in p and m<*k, or m is a sibling of m1, or m is a descendant of a sibling of m1, where m1≤*n}. It is fixed at the time a node is first selected and remains unchanged. The ancestry(k) is further divided into 3 sets: the ancestors(k) (nodes ≤*k), the residue(k) (siblings of ancestors of k in unclosed branches) and the closed ancestry(k) (siblings of ancestors of k and their descendants in closed branches)). A three-dimensional tableau is closed if at least one of its planes is closed.

C: The semi-linear refinement for clauses develops a three-dimensional tableau by applying the following rules:

(a) The top clause that begins the tableau is chosen to be a clause that is necessary for all refutations that are to be found. Its nodes belong to plane 1.

(b) Open branches in a plane are selected for completion in a left-right order.

(c) Within a selected branch in a selected plane there are four possible steps:

(ci) A leaf literal is selected that will close the branch by unification.

(cii) A leaf literal L at node k in plane p is selected for the first time and an input clause C is selected such that it contains a (leftmost) literal that is complementary to L. The clausal tableau rule is applied to C at node k. The new nodes are labelled with plane p.

(ciii) A literal L at node k that has been selected before is selected again. It is likely that P(k) is not a singleton.

Either: a new plane p' is created in which L is the leaf literal of the selected branch and the branch may be closed. All nodes in the ancestry of k and in the same planes P as k are updated by p' : ∀n ∈ ancestry(k) : P(n) ⊇P . P(n) becomes P(n) ∪ {p'}.

Or: a new plane p' is created using the clausal rule below k using an input clause with a (leftmost) literal complementary to L. The new nodes are in plane p' and nodes in the ancestry(k) and in P are again updated by p'.

(civ) Two planes are combined: a leaf literal L at node k in plane p is selected and another literal complementary to L at node m in plane p' is selected. A copy of nodes in residue(m) in plane p' is made below k. The new nodes are in p. Any copied node in the residue(m), labelled by a literal J and which also occurs in residue(k) such that the instance of J in both planes p and p' is identical, must be closed by merging; i.e. the closure below J' can be made below J as well.

Notes:
(i) To avoid duplication of nodes in a branch, there is a restriction on operation(civ).
(ii) (civ) may also give rise to a new plane if L is not a leaf node.
(iii) It is both desirable and possible to share the residue(n) between planes when the substitutions of the two planes results in common literals. For example, if a clause $P(x) \vee B \vee C(x)$ is developed in a tableau and P(x) is extended in two different ways, making respective substitutions x=a, x=b, then B can be shared but C(x) cannot. Anyway, nodes in closed ancestry(m) only need to be implicitly copied. The various details are in [Broda].

The procedure outlined above is a very flexible one and additional restrictions can be imposed. For example, (civ) could only be allowed when all nodes in residue(m) in p' unify with their counterparts in the residue(k) in p. The above procedure includes as a special case simple linear refutations in which only one plane is ever developed. Hence its completeness is assured, in the sense that a refutation of the correct form exists, because such linear tableaux refutations exist. We also have :

Theorem: Any ground tableau that satisfies a chain property can be lifted in the spirit of the semi-linear refinement; i.e. common structures can be shared.

Informally, the *chain property* holds when linearly developed sequences of clause instances of clauses in a set C, below a node k, are used in the same relative order.

Because of the plane combination operation, it is now unclear exactly what constitutes the search space. For example, what begins as two different branches of the search space turns into one branch after two planes are combined. This problem can be overcome by incorporating connection graph book-keeping (cgbk) [Kowalski]. For the semi-linear refinement we have:

Theorem:
Any ground tableau that satisfies the chain property and the properties: (a) the leftmost literal of every clause closes a branch (b) all instances of the selected top clause have the new literal G disjoined and the root node is labelled by $\neg G$ (c) no literal occurs in a branch more than once, can be lifted such that if the steps are followed exactly and cgbk is used a closed tableau results.

This is not the property we really want, which is that any ground tableau formed with the above properties can be lifted so that, in whatever order the steps are taken and if cgbk is used, a closed tableau results. For details of the difficulties see [Broda].

If the sentences are not all clauses then it is possible to include a compilation phase that reduces sentences to a generalised clausal form, such that on each branch arising from developing a sentence there are only literals or universally quantified literals. The semi-linear refinement can then be extended to the new situation.

References

Broda, K The Application of Semantic Tableaux with Unification to Automated Deduction, PhD Thesis, Dept. Computing, Imperial College, (1991)

Kowalski, R. A. A Proof Procedure Using Connection Graphs, JACM, Vol. 22, No. 4 (1975)

Search Space Pruning by Checking Dynamic Term Growth

Stefan Brüning*
FG Intellektik, FB Informatik, Technische Hochschule Darmstadt
Alexanderstraße 10, D–6100 Darmstadt (Germany)
E-mail: stebr@intellektik.informatik.th-darmstadt.de

Abstract

In this paper we present a method to detect non-terminating or failing queries based on analyzing the dynamic growth of terms. It overcomes restrictions known from approaches to preclude infinite loops in the field of logic programming. The general idea is to predetermine what may happen to a term performing inference steps. Various well-known techniques and results of the theory of formal languages are used. The strength of our technique is emphasized by the fact that using it we are able to decide Horn-formulas consisting of facts, two-literal clauses, and a goal literal all of them restricted to unary predicate and unary function symbols.

1 Introduction

One of the main goals of the research in the field of automated theorem proving is to develop proof methods which are on the one hand as general as possible and on the other hand as adequate[1] as possible. General methods, like the resolution principle [Rob65], analytic tableaux [Bet55, Smu71], and the connection method [Bib87, Bib92] are well known. But calculi developed from these methods seem only to be adequate to some extent if they are augmented by techniques to control the proof process and especially to keep the search space as small as possible.

In this paper we will address this problem by proposing techniques to detect failing or non-terminating queries. Several approaches concerning this problem have been presented in the past. In the field of logic programming various techniques for providing logic programming interpreters with a capability to preclude some infinite loops at runtime were examined. Roughly speaking, most of them (for example cf. [Bes89], [ABK89], [Smi86]) are based on subsumption tests between subgoals occurring during an SLD-derivation. The main interest of these approaches is to define complete and sound loop checks, i.e. loop checks that prune every infinite SLD-derivation in such a way that no answer substitution for a goal is lost. Generally, this cannot be achieved without much effort. In [ABK89] it is shown that a loop check taking only the actual derivation into account cannot be complete and sound, even for logic programs without function symbols.

The major lack of loop checks based on subsumption is that they are not applicable in those situations where subsumption relations between subgoals cannot occur. For example, such techniques are not applicable — except for the detection of identical subgoals — if the subgoals under consideration are ground. Furthermore, since they were developed in the field of logic programming most of them only work properly on Horn-formulas and

*The author is supported by the Deutsche Forschungsgemeinschaft (DFG) within project KONNEKTIONSBEWEISER under grant no. Bi 228/6-1.

[1] We will consider a technique as being adequate if, roughly speaking, it solves simpler problems faster than more difficult ones.

therefore cannot be taken over easily to the more general field of automated theorem proving. Only for some special cases this is possible without any modifications. A well known example is the *identical ancestor pruning* rule as realized in the theorem provers SETHEO [LSBB92] or PTTP [Sti88].

Some researchers prefer a static approach to avoid infinite loops by analyzing a program at compile time. An example of such an approach is cycle unification [Wür92, Sal92]. Answers are given to the question of how many iterations through a cycle of the form $P(l_1,\ldots,l_n) \leftarrow P(r_1,\ldots,r_n)$ have to be considered given a fact $P(t_1,\ldots,t_n)$ and a query $\leftarrow P(s_1,\ldots,s_n)$. This is achieved by studying variable dependencies. The problem which occurs is that even for this small class of problems the question is undecidable in general if the cycle is unrestricted, i.e. $t_1,\ldots,t_n,s_1,\ldots,s_n$ are arbitrary terms (cf. [HW92]).

The approach presented in this paper is based on the analysis of growth and elimination of function symbols caused by unification during the proof process. The general idea is to predetermine what may happen to a term performing inference steps. We define techniques to detect failing or non-terminating queries which are applicable to arbitrary subgoals, including those which cannot be treated by subsumption tests. Our techniques work on arbitrary Horn-formulas. Thus, we overcome main problems of pruning techniques based on the methods mentioned above. Furthermore, a generalization to non-Horn formulas is possible. However, up to now our approach lacks some desired properties. We cannot take dependencies into account in order to detect clashes or performing occur-checks. Nevertheless, the techniques developed are very strong. This is emphasized by the fact that using one of them we are able to decide Horn-formulas containing facts, two-literal clauses, and a goal literal all of them restricted to unary predicate and unary function symbols. For more general classes the techniques loose strength, but remain useful.

The following example will illustrate our idea.

Example 1.1 *Consider the logic program*

$$\begin{array}{lll} P(f(a)). & & \\ P(x) & \leftarrow & P(f(x)). \\ P(f(x)) & \leftarrow & P(f(f(f(x)))). \end{array}$$

together with the goal clause $G = \leftarrow P(a)$. *A standard proof procedure — for example a PROLOG interpreter — would be able to refute* G *performing a resolution step with the first rule and resolving the resolvent* $\leftarrow P(f(a))$ *with the fact yielding the empty clause. However, a two-fold application of the first rule or an application of the second rule would result in the subgoal* $\leftarrow P(f(f(a)))$ *which leads to an infinite loop. A standard PROLOG interpreter is not able to detect such a loop. This is crucial, since a simple reordering of the rules and the fact of the above program would lead a standard PROLOG interpreter only to use the rules resulting in an infinite loop.*

However, if we consider which function symbols are eliminated or added using the rules we are able to find out whether a derivation succeeds. Applying the first rule, the term substituted for x *is extended by the functor* f. *Applying the second rule, the functor* f *of the term occurring in a subgoal of the form* $\leftarrow P(f(t))$ *is eliminated by unifying with* $f(x)$. *But in the body of the rule the resulting term* t *growths to* $f(f(f(t)))$. *Thus, if a subgoal* $\leftarrow P(t)$ *has to be refuted, using one of the rules would lead to the growth of the term* t *to* $f(t)$ *or* $f(f(t))$ *in the resulting subgoal. Therefore, in order to deduce the empty clause via the fact* $P(f(a))$,*at most one application of the first rule has to be considered.*

Most techniques presented in this paper are based on the application of techniques known from the field of formal languages. Therefore we assume the reader to be familiar

with the classes of formal languages contained in the Chomsky-hierarchy (cf. [HU69]). Especially, we will use regular and context-free languages and the corresponding acceptor models, the finite automata and non-deterministic push-down acceptors. The definitions needed in this paper will be given in section 2.

The paper is organized as follows. In section 2 we introduce definitions and notations. In section 3 we present a first pruning technique based on the analyses of prefix-elimination represented by position graphs. In section 4 we combine the aspects of elimination and growth of functors to yield a stronger result. Using the technique presented there we are able to decide Horn-formulas consisting of facts, two-literal clauses, and a goal literal all of them restricted to unary predicate and unary function symbols. Throughout section 3–4 we restrict ourself to Horn formulas restricted to unary function. In section 5 we overcome this restriction. We further sketch how to include dependencies between terms. In section 6 we conclude.

Due to the lack of space proofs of theorems and corollaries are omitted. They can be found in [Brü92a].

2 Preliminaries

In this section we present definitions and notations which we need throughout this paper. We assume the reader to be familiar with the concepts of first-order logic. The terms "(first-order) Horn formula" and "logic program" will be used synonymously.

In what follows we define the concepts of a prefix, prefix-elimination, and prefix-growth.

Definition 2.1 *Prefix of a term: If t is a term of the form $f_1(f_2(\ldots f_n(t')))$ where $f_1, \ldots, f_n$ denote unary function symbols, we call $f_1 \ldots f_n$ a* prefix *of t. $f_1 \ldots f_n$ is the* maximal *prefix of t if t' is either a variable or a constant.*

Definition 2.2 *Prefix-elimination: A prefix $t = t_1 \ldots t_n$ occurring in a (sub)goal literal G can be eliminated by a program $\mathcal{P}$ if after performing a finite number of inference steps the (explicit instances of the) functors $t_1, \ldots, t_n$ do not occur in the resulting subgoals any more.*

Definition 2.3 *Prefix-growth: A prefix $t = t_1 \ldots t_n$ occurring in a (sub)goal literal G grows performing inference steps if a prefix of the form $s_1 \ldots s_m t_1 \ldots t_n$ (where the explicit instances of the functors $t_1, \ldots, t_n$ are preserved) occurs in the resulting subgoals. We say the prefix $s_1 \ldots s_m$ is* added *to t.*

The concepts of prefix-elimination and prefix-growth are illustrated by the following example.

Example 2.1 *Consider the logic program*

$$\begin{array}{lcl} P(a). & & \\ P(f(x)) & \leftarrow & P(x) \\ P(x) & \leftarrow & P(g(x)) \end{array}$$

together with the goal clause $\leftarrow P(f(f(x)))$. The prefix ff occurring in the literal $\leftarrow P(f(f(x)))$ may be eliminated performing two resolution steps with the first rule. The application of the second rule would lead to a growth of the prefix ff resulting in gff.

We further use the following terminology. With $\mathcal{P}$ we always denote a logic program. $F_H^{\mathcal{P}}$ is the set of function symbols occurring in the head literals of the rules of $\mathcal{P}$. In the same way $F_B^{\mathcal{P}}$ denotes the set of function symbols occurring in the bodies of the rules of $\mathcal{P}$. The set $(F_B^{\mathcal{P}})^{-1}$ is defined as $\{f^{-1} | f \in F_B^{\mathcal{P}}\}$. For simplicity, the exponent $\mathcal{P}$ is omitted if it is obvious which logic program is under consideration.

Definition 2.4 *Finite automata: A non-deterministic finite automata (FA) $\mathcal{F}$ is defined by a 5-tuple $(\Sigma_{\mathcal{F}}, K_{\mathcal{F}}, S_{\mathcal{F}}, F_{\mathcal{F}}, \delta_{\mathcal{F}})$ where $\Sigma_{\mathcal{F}}$ is the input alphabet, $K_{\mathcal{F}}$ the set of states, $S_{\mathcal{F}} \in K_{\mathcal{F}}$ the initial state, and $F_{\mathcal{F}} \subseteq K_{\mathcal{F}}$ the set of final states of $\mathcal{F}$. $\delta_{\mathcal{F}}$ is a relation $\Sigma_{\mathcal{F}}^* \times K_{\mathcal{F}} \mapsto 2^{K_{\mathcal{F}}}$ which represents the non-deterministic program of $\mathcal{F}$. By $\mathcal{L}_{\mathcal{F}}$ we denote the regular language determined by $\mathcal{F}$. $\mathcal{L}_{\mathcal{F}}^1$ is the regular language determined by $(\Sigma_{\mathcal{F}}, K_{\mathcal{F}}, S_{\mathcal{F}}, \{f\}, \delta_{\mathcal{F}})$ where $f \in F_{\mathcal{F}}$ must hold.*

For each non-deterministic finite automata $\mathcal{F}$ there exists a deterministic finite automata $\mathcal{F}'$ such that $\mathcal{L}_{\mathcal{F}} = \mathcal{L}_{\mathcal{F}'}$ holds. Therefore, we do not distinguish these acceptor models throughout this paper.

A finite automata can be represented by a directed graph $(\mathcal{V}, \mathcal{E})$[2]. The set of nodes $\mathcal{V}$ corresponds to the set of states of the finite automata. Consequently, one element of $\mathcal{V}$ is marked as *start node* and a subset of $\mathcal{V}$ is marked as *end nodes*. If $(p, w, q) \in \delta_{\mathcal{F}}$ for $w \in \Sigma_{\mathcal{F}}^*$ holds, there exists an edge from the node representing p to the node representing q labelled with w.

In what follows ϵ denotes the empty word.

3 Pruning search paths analyzing prefix-elimination

In this section we will introduce a first technique to detect failing or non-terminating queries. For the moment we will focus our attention on prefix-elimination. For simplicity, we restrict ourself to the class of Horn-formulas containing function symbols with at most one argument. As we will demonstrate in section 5 our approach can be generalized to work on formulas containing arbitrary function symbols. The elimination of prefixes can be represented by position graphs.

Definition 3.1 *Position graph: Let $\mathcal{P}$ be a logic program. A position graph $PG_{\mathcal{P}}$ of $\mathcal{P}$ is a directed graph whose nodes are argument positions of literals appearing in $\mathcal{P}$. Let p and q be argument positions occurring in different literals P and Q, respectively. (p, q) is an edge of $PG_{\mathcal{P}}$ if the term occurring in p is not ground and one of the following conditions holds:*

c-edge: *(P, Q) is a weakly unifiable connection[3]. Q is the head literal of a clause and p and q denote the same position in the literals involved in this connection.*

nc-edge: *Let P be the head literal of a rule and Q a non-pure[4] literal in its body. The variables occurring in p and q are the same.*

[2] As usual a graph is represented by a set $\mathcal{V}$ of nodes and a set $\mathcal{E}$ of edges where an edge is a tuple of the form (p, q) with $p \in \mathcal{V}$ and $q \in \mathcal{V}$.

[3] A connection is an unordered pair of literals with the same predicate symbol and different sign. A connection (P, Q) is *weakly unifiable* iff $\neg P$ and Q' are unifiable where Q' emerges from Q by substituting each variable by a fresh one

[4] A literal is *pure* if it is not possible to resolve it with another literal occurring in the corresponding program.

The edges are labeled in the following way.

$$l(p,q) = \begin{cases} \{f_i \dots f_n \mid 1 \leq i \leq n\} \cup \{\epsilon\} & (p,q) \text{ is a nc-edge, } f_1, \dots, f_n \\ & \text{is the maximal prefix occurring in } p \\ \{\epsilon\} & \text{else} \end{cases}$$

Furthermore, for a node p of $PG_{\mathcal{P}}$ we define the sets $in(p)$ and $out(p)$ as:

$$\begin{aligned} in(p) &= \{p' \mid (p',p) \text{ is a c-edge or a nc-edge of } PG_{\mathcal{P}}\} \\ out(p) &= \{p' \mid (p,p') \text{ is a c-edge or a nc-edge of } PG_{\mathcal{P}}\} \end{aligned}$$

The labels of nc-edges of a rule R describe the prefix-elimination caused by this rule. Let G be the current subgoal and t the prefix of a term occurring in G. We assume that R is used during the derivation of G. If w_1w_2 is the maximal prefix of a term occurring in the head of R then w_2 may represent the elimination of a part of t whereas w_1 may represent the elimination of a prefix that was added to t performing some inference steps. Because we do not consider prefix-growth we have to include each postfix of a maximal prefix occurring in the head of R into the labels.

Example 3.1 *Consider the following logic program $\mathcal{P}$*

$$\begin{array}{lcl} P(a). & & \\ P(fg(x)) & \leftarrow & P(x) \\ P(x) & \leftarrow & P(f(x)) \end{array}$$

together with the query $G = \leftarrow P(g(x))$. The position graph of $\mathcal{P} \cup \{G\}$ contains an nc-edge between the positions occurring in the first rule which is labelled with the set $l = \{fg, g, \epsilon\}$. All other edges are labelled with $\{\epsilon\}$. Since we do not consider prefix-growth explicitly we have to build l in this way. Only considering the elimination of the prefix fg would not be sufficient. We also have to consider the elimination of g since this prefix may be eliminated after being extended by the prefix f (caused by the second rule).

In some special cases (cf. [Brü92a]) it is possible to label the edges in a way that the process of prefix elimination is modelled more exactly.

If $H \leftarrow B_1, \dots, B_n$ is a clause in $\mathcal{P}$ there may exist more than one nc-edge of the form (p, q_i). Here, p denotes a term position in H whereas q_i denotes a term position in B_i ($1 \leq i \leq n$). To eliminate the prefix of a term which is substituted for the variable occurring in p during a derivation it is necessary to eliminate this prefix during the derivation of each subgoal $\leftarrow B_i$ where an nc-edge (p, q_i) of $PG_{\mathcal{P}}$ exists. But to recognize that the prefix *cannot* be eliminated it is sufficient to show that it cannot be eliminated during the attempt to refute only one of the subgoals $\leftarrow B_i$. This leads us to the definition of reduced position graphs. This definition requires the definition of p-reachability.

Definition 3.2 *p-reachability: Let $\mathcal{P}$ be a logic program and let p and q be nodes of $PG_{\mathcal{P}}$. q is* p-reachable *if there exists a sequence of edges*

$$(p, r_1)(r_1, r_2) \dots (r_{n-1}, r_n)(r_n, q)$$

from p to q where (p, r_1), (r_n, q), and (r_i, r_{i+1}) $(1 \leq i \leq n-1)$ are edges of $PG_{\mathcal{P}}$

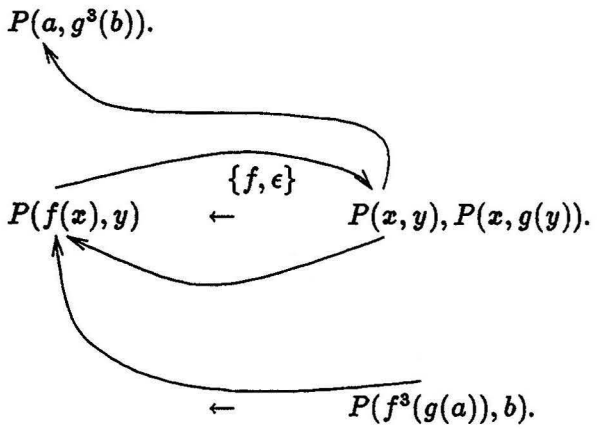


Figure 1: A reduced position graph

Definition 3.3 *Reduced position graph: Let $\mathcal{P}$ be a logic program, G a goal literal and p a position occurring in G. A* reduced position graph $RPG^p_{\mathcal{P}\cup\{G\}}$ *of $\mathcal{P} \cup \{G\}$ w.r.t. p is a subgraph of $PG_{\mathcal{P}\cup\{G\}}$ satisfying the following conditions:*

1. *Every node of $RPG^p_{\mathcal{P}\cup\{G\}}$ is p-reachable*
2. *Every c-edge (p', q) of $PG_{\mathcal{P}\cup\{G\}}$ is an c-edge of $RPG^p_{\mathcal{P}\cup\{G\}}$ if p' is p-reachable*
3. *Let p' be a p-reachable node occurring in the head-literal of a clause. Then, only one nc-edge (p', q) of $PG_{\mathcal{P}\cup\{G\}}$ has to be an nc-edge of $RPG^p_{\mathcal{P}\cup\{G\}}$*

Example 3.2 *The reduced position graph shown in figure 1 is a $RPG^p_{\mathcal{P}\cup\{G\}}$ of the following logic program $\mathcal{P}$ together with the goal clause $G = \leftarrow P(f^3(g(a)), b)$ w.r.t. the first position of G. For simplicity we omit the labels of c-edges.*

$$\begin{array}{lll} P(a, g^3(b)). & & \\ P(f(x), y) & \leftarrow & P(x, y), P(x, g(y)). \end{array}$$

The "flow" of a prefix in a reduced position graph corresponds to the "flow" of this prefix during a subproof in which only one relevant subgoal per rule is considered. As we will see, the regular language $\mathcal{L}$ which corresponds to a reduced position graph — if we mark p as start node and every node q of $RPG_{\mathcal{P}\cup\{G\}}$ with $in(q) \neq \emptyset$ and $out(q) = \emptyset$ as end node — represents the set of prefixes which can be eliminated in this subproof. However, to eliminate a prefix one has to consider the subproofs of each relevant subgoal of a clause.

The following theorem provides a basis to prune search paths using reduced position graphs. It states under which circumstances a prefix of a term occurring in a certain position in a (sub)goal cannot be eliminated. Note that we have to take into account whether this term contains a variable. In this case, we must consider possible instantiations.

Theorem 3.1 *Let $\mathcal{L}_\mathcal{F}$ be a regular language where $\mathcal{F}$ is the finite automata determined by a reduced position graph of a program $\mathcal{P}$ and a goal literal G w.r.t. a position p. Let t' be the maximal prefix of the term t occurring in G at position p.*

If t is ground t' cannot be eliminated using the rules of $\mathcal{P}$ if $t' \notin \mathcal{L}_{\mathcal{F}}$ holds. Otherwise, if t contains a variable t' cannot be eliminated if $t's \notin \mathcal{L}_{\mathcal{F}}$ for an arbitrary prefix $s \in (F_H)^*$ holds.

The following corollary is a consequence of theorem 3.1. It defines a technique to detect nonterminating or failing queries based on the analysis of prefix-elimination.

Corollary 3.1 *Let $\mathcal{F}$ be the FA determined by $RPG^{p}_{\mathcal{P}\cup\{G\}}$ where $\mathcal{P}$ is a logic program and G a (sub)goal literal. Let t be a term occurring in G at position p. If t is ground and the first of the following conditions holds, G cannot be refuted. Otherwise, if t contains a variable G cannot be refuted if both conditions hold.*

- *There exists no term t_2 occurring in an end node corresponding to a finite state q of $\mathcal{F}$ such that t is unifiable with t_1t_2 where t_1 denotes a prefix which is element of the regular language $\mathcal{L}^{q}_{\mathcal{F}}$.*
- *$t_1(F_H)^* \cap \mathcal{L}_{\mathcal{F}} = \emptyset$, where t_1 denotes the maximal prefix of t.*

Example 3.3 *Recall the logic program given in example 3.2 augmented by an additional rule*

$$P(x, y) \leftarrow P(f(x), y).$$

Clearly, an attempt to refute the goal $\leftarrow P(f^3(a), b)$ would lead to an infinite loop using this rule. Applying the technique given by corollary 3.1 we are able to detect this as follows. The node occurring in the fact containing the term a is marked as sole end node and the node containing the term $t = f^3(g(a))$ is marked as start node. The corresponding finite automata represents the regular language $\{f^i \mid i \geq 0\}$. To unify a subgoal with the fact, the whole prefix of t has to be eliminated. But this is impossible since $f^3g \notin \{f^i \mid i \geq 0\}$ holds.

4 Introducing prefix-growth

In the preceding section we only considered prefix-elimination neglecting prefix-growth. In this section we will sketch how these two aspects of manipulating a term during a proof can be combined. This will give us much stronger techniques to prune search paths. The following example will illustrate the ideas presented in this section.

Example 4.1 *Let us reconsider the logic program presented in example 1.1*

$$\begin{array}{lll} P(f(a)). & & \\ P(x) & \leftarrow & P(f(x)). \\ P(f(x)) & \leftarrow & P(f(f(f(x)))). \end{array}$$

together with the goal clause $G = \leftarrow P(f^i(a))$. The pruning technique presented in the previous section does not detect that G cannot be refuted if $i \geq 2$ holds. If prefix-growth is taken into account it is easy to see that applying one of the rules to a subgoal of the form $\leftarrow P(f^i(a))$ yields a subgoal $\leftarrow P(f^{i+j}(a))$ with $j \in \{1, 2\}$. Since the only fact contains the ground term $f(a)$ the empty clause cannot be deduced from a subgoal of the form $\leftarrow P(f(f(\ldots)))$.

Again, we use position graphs for our purposes. We modify definition 3.1 in the following way. If $(p,q) \in PG_{\mathcal{P}}$ is an nc-edge and t_1 denotes the maximal prefix occurring at p and t_2 the maximal prefix occurring at q then $l(p,q) = \{t_1 t_2^{-1}\}$ The definition of a restricted position graph is exactly the same as in the preceding section w.r.t. to the new labels.

The idea underlying this form of labels is the following. Consider the label $t_1 t_2^{-1}$ of an nc-edge of a rule R where t_1 and t_2 denote arbitrary prefixes. Then, R must be of the form:

$$H(\ldots, t_1(x), \ldots) \leftarrow \ldots, B(\ldots, t_2(x), \ldots), \ldots$$

If R is applied to a (sub)goal literal L a term t occurring in L has to be unified with $t_1(x)$. As in the last section t_1 represents the prefix which is eliminated from t. t_2 represents the prefix which is added afterwards to the term substituted for x. In that way we can represent the growth and the elimination of a prefix caused by the application of R.

A word w of the regular language $\mathcal{L}$ determined by a (reduced) position graph can be transformed by the following rule:

$$t^{-1}t = \epsilon \tag{1}$$
$$(t_1 \ldots t_n)^{-1} = t_n^{-1} \ldots t_1^{-1} \tag{2}$$

The first rule describes that the generation and elimination of a prefix yields the identity. The second rule is an immediate consequence of the first one.

For our context it is important that only words have to be considered if they can be reduced by (1) and (2) to a word of the form $f_1 \ldots f_n g_1^{-1} \ldots g_m^{-1}$. Suppose a word $w \in \mathcal{L}$ cannot be transformed to this form. Then, w contains at least one subword of the form $g^{-1}f$ with $g \neq f$. Thus, the function symbol g is generated and afterwards the function symbol f is eliminated. If we consider our context, that means we want to unify a term of the form $g(\ldots)$ with a term of the form $f(\ldots)$.

It remains to show that, given a word w, it is decidable whether there exists a word $w' \in \mathcal{L}$ such that w' can be transformed to w by the rules (1) and (2). This can be shown by building a non-deterministic push-down-acceptor (PD). A construction is given in [Brü92a]. Thus, given a finite automata $\mathcal{F}$ represented by a (reduced) position graph we are able to define a push-down acceptor which accepts the words of $\mathcal{L}_{\mathcal{F}}$ transformed by the rules (1) and (2).

In what follows $\mathcal{L}^q_{\mathcal{PD}}$ denotes the context-free language determined by the push-down acceptor $\mathcal{PD}$ where the set of final states $F_{\mathcal{F}}$ of the underlying finite automata $\mathcal{F}$ consists only of the state q.

Like in the previous section we can prove the following theorem which provides a technique to detect non-terminating or failing queries.

Theorem 4.1 *Let t be a term occurring in a (sub)goal literal G at position p.*

Let $\mathcal{F}$ be the finite automata determined by a reduced position graph $RPG^p_{\mathcal{P}\cup\{G\}}$. Furthermore, let t_q be the term occurring in an end node corresponding to a finite state $q \in F_{\mathcal{F}}$. Let $\mathcal{PD}$ be the PD determined by $\mathcal{F}$.

We distinguish two cases. If t is ground or we assume that not the whole prefix of t can be eliminated, then if there exists no $q \in F_{\mathcal{F}}$ such that properties (1a), (2a), and (3) are fulfilled, the subgoal cannot be refuted using $\mathcal{P}$. Otherwise, if we assume that the whole prefix of the non-ground term t was eliminated and there exists no $q \in F_{\mathcal{F}}$ such that the properties (1b), (2b), and (3) are fulfilled, the subgoal cannot be refuted using $\mathcal{P}$.

(1a) $t = t_1 t_2$

(1b) $t = t_1(x),\ t_2 = x$

(2a) $\exists t_3 :\ t_1 t_3^{-1} \in \mathcal{L}^q_{\mathcal{PD}}$

(2b) $\exists t_3, t_4 :\ t_1 t_4 t_3^{-1} \in \mathcal{L}^q_{\mathcal{PD}}$

(3) $t_3 t_2$ *is unifiable with* t_q

Here, t_4 represents the prefix of a possible instantiation of the variable x.

To apply the theorem the remaining problem is to find suitable instantiations for t_3. These instantiations can be determined by taking the prefix of t_2 and the prefixes occurring in the end nodes of the reduced position graph into consideration. For more details we have to refer to [Brü92a].

4.1 A decidable class of formulas

Applying the technique presented in this section we are able to decide Horn-formulas consisting of facts, two-literal clauses, and a goal literal all of them restricted to unary predicate and unary function symbols. Although this is no new result[5] it emphasizes the strength of our technique especially if unary function symbols are considered.

Theorem 4.2 *Let $\mathcal{P}$ be a logic program consisting of facts, two literal clauses, and a goal literal G all of them restricted to unary predicate and function symbols.*

G can be refuted by $\mathcal{P}$ iff the conditions of theorem 4.1 hold.

5 Generalizations

In this section we will present several generalizations of the techniques presented in the previous sections. Due to the lack of space we can only sketch the underlying ideas and cannot present the technical details. They can be found in [Brü92a, Brü92b].

5.1 Generalization to arbitrary function symbols

The techniques proposed in the previous sections were only defined for logic programs using functions with at most one argument. In this subsection we will sketch how they can be lifted to work on arbitrary Horn-formulas. We can take over the technique presented in the previous section in an easy fashion if we do not consider how a whole term is modified applying rules on (sub)goals but only consider changes on infixes of these terms.

Definition 5.1 *Infix of a term: Let t be a term of the following structure:*

$$\begin{aligned} t &= f_1(\ldots, t_1, \ldots) \\ t_1 &= f_2(\ldots, t_2, \ldots) \\ &\vdots \\ t_{n-1} &= f_n(\ldots, t_n, \ldots) \end{aligned}$$

[5] Rabin [Rab69] showed that the monadic second order logic for n successor functions is decidable. This result implies decidability for first order logic restricted to unary predicate and function symbols.

We call $f_1 \ldots f_n$ an infix *of t and the term $f_1 \ldots f_n(t_n)$ an* infix term *of t. $f_1 \ldots f_n$ is the* maximal *infix of an infix term $f_1 \ldots f_n(t')$ if t' is either a variable or a constant.*

Example 5.1 *Let $t = f(g(a,b), h(f(x,y)))$. Then, $\{fg, fhf\}$ is the set of maximal infixes of t. The set $\{fg(a), fg(b), fhf(x), fhf(y)\}$ contains the infix terms of t.*

In the same way we defined position graphs, we can define infix-position graphs which represent the modifications of infix-terms caused by the application of the rules of a given logic program. Based on infix-position graphs a theorem is provable which is similar to theorem 4.1.

5.2 Generalization to full first-order logic

A generalization to handle full first-order formulas is possible too. Then, we have to consider that each literal of a clause representing a rule might be used as head literal of the clause. Thus, the flow of a prefix through a clause is not determined only in one direction, i.e. in the Horn case from a position in the unique head of a rule to a position occurring in a literal contained in its body.

All other techniques mentioned in this paper — except some special cases of subsumption tests — are defined to work on Horn-formulas and are therefore not applicable in the more general field of automated theorem proving without any modification.

5.3 Analyzing dependencies

Sometimes it is necessary to take dependencies between term positions into account in order to detect non-terminating or failing queries. This can be achieved by incorporating identifiers for the rules into labels of nc-edges. In this way we can control how — i.e. by using which rules — a certain word of the language determined by a position graph is generated. The following example illustrates this idea:

Example 5.2 *Consider the following logic program*

$$
\begin{array}{lcl}
Q(a,a). & & \\
Q(x,f(y)) & \leftarrow & Q(x,y). \\
P(a,a). & & \\
P(f(x),y) & \leftarrow & P(x,y). \\
R(x,y) & \leftarrow & R(f(x),f(y)). \\
R(x,y) & \leftarrow & Q(x,y). \\
R(x,y) & \leftarrow & P(x,y).
\end{array}
$$

together with the goal $G = \leftarrow R(f(a),f(a))$. The techniques presented in this paper do not detect that the attempt to refute the goal must lead to an infinite loop. This is, because the first term of G can be eliminated using the second and fifth rule whereas the second term can be eliminated using the first and fourth rule. However, if the applications of the rules are considered explicitly it is possible to detect that these transformations cannot be done simultaneously.

6 Conclusion

In this paper we presented techniques to detect failing or non-terminating queries. These techniques are based on the general approach of analyzing the prefix-growth and prefix-elimination during the proof process. In this way it is possible to determine what may happen to a term performing inference steps. Our approach is suitable to work on arbitrary first-order formulas. To demonstrate its power it is shown that using the technique presented in section 4 the class of Horn-formulas consisting of facts, two-literal clauses, and a goal literal all of them restricted to unary predicate and function symbols is decidable.

We have to major applications of the presented techniques in mind. Firstly, they can be used at compile time to detect (sub)goals that can never be refuted. However, our techniques sometimes are not able to realize at compile time that a goal cannot be refuted. But maybe after performing some inference steps they are able to do this.

Example 6.1 *Consider the following logic program.*

$$\begin{array}{lll} P(f^4(a)). & & \\ P(x) & \leftarrow & P(f(x)), P(f(f(x))). \end{array}$$

Because we only consider one relevant subgoal per rule in a reduced position graph we cannot recognize that the goal $\leftarrow P(a)$ *leads to an infinite loop. If the technique presented in section 4 is used dynamically it will detect that no solution can be found. After performing three inference steps, the subgoal* $P(f^6(a))$ *is generated which cannot be refuted according to the technique.*

Therefore, our techniques should also be used at runtime. But this requires some more considerations because performing one of them after each inference step would lead to an enormous overhead. Heuristics are needed that give hints when our techniques should be applied. For example, they might become useful if the depth of terms occurring in a derivation or the number of used instances of a rule exceed a certain limit. Furthermore, heuristics should be developed to indicate which reduced position graphs should be examined. To examine all possible reduced position graphs might be too expensive. Such heuristics can depend on the actual prefixes under consideration. For example, if one wants to check whether a prefix containing several occurrences of the functor f can be eliminated, those reduced position graphs should be considered which contain fewest occurrences of labels indicating the elimination of f.

It is worth emphasizing that our techniques lead to a reduction of the search space only if a goal *cannot* be refuted. For example, it does not restrict the application of a rule of the form

$$P(x, y) \leftarrow P(y, x)$$

if a solution to refute the occurring subgoals exists. Here, techniques based on subsumption or cycle unification have to be used.

Other applications of our approach to analyze prefix-elimination and prefix-growth are conceivable. For example, as sketched in [Brü92a], it is sometimes possible to determine an upper bound for the number of used instances of a rule during a derivation.

References

[ABK89] K. R. Apt, R. N. Bol and J. W. Klop: *On the safe termination of PROLOG programs*, Proc. 6th Conf. on Logic Programming, G. Levi and M. Martelli (eds.), MIT Press, Cambridge MA, 1989.

[Bes89] P. Besnard: *On infinite loops in logic programming*, IRISA, Publication Interne No. 488, 1989.

[Bet55] E. W. Beth: *Semantic entailment and formal derivability*, Mededlingen der Koninklijke Nederlandse Akademie van Wetenschappen, 18(13), 1955.

[Bib87] W. Bibel: *Automated Theorem Proving*, Vieweg Verlag, Braunschweig, second edition, 1987.

[Bib92] W. Bibel: *Deduktion, Automatisierung der Logik*, Oldenbourg Verlag, München, 1992.

[Brü92a] S. Brüning: *Search Space Pruning by Checking Dynamic Term Growth*, Technical Report AIDA-92-19, TH Darmstadt, 1992.

[Brü92b] S. Brüning: *On Search Space Pruning by Checking Dynamic Term Growth*, Technical Report, TH Darmstadt, forthcoming.

[HU69] J. E. Hopcroft and J. D. Ullman: *Formal languages and their relation to automata*, Addison-Wesley Publ.Co., (Series in Computer Science and Information Processing), 1969.

[HW92] P. Hanschke and J. Würtz: *Satisfiability of the smallest binary program*, to appear.

[LSBB92] R. Letz and J. Schumann and S. Bayerl and W. Bibel: *SETHEO: A High-Performance Theorem Prover*, Journal of Automated Reasoning 8, 1992.

[Rab69] M. O. Rabin: *Decidability of second order theories and automata on infinite trees*, Trans. Amer. Math. Soc. 141, 1969.

[Rob65] J. A. Robinson: *A machine-oriented logic based on the resolution principle*, Journal of the ACM, 12(1), 23–41, 1965.

[Sal92] G. Salzer: *Solvable Classes of Cycle Unification Problems*, Proc. IMYCS'92, 1992.

[Smi86] D. E. Smith and M. R. Genesereth and M. L. Ginsberg: *Controlling recursive inference*, Artificial Intelligence, 30, 343–389, 1986.

[Smu71] R. M. Smullyan: *First-Order Logic*, Ergebnisse der Mathematik und ihrer Grenzgebiete, Springer-Verlag, 1971.

[Sti88] M. E. Stickel: *A PROLOG technology theorem prover: implementation by an extended PROLOG compiler*, Journal of Automated Reasoning, 4, 353–380, 1988.

[Wür92] J. Würtz: *Unifying Cycles*, research report RR-92-22, DFKI, Saarbrücken, 1992.

Decidable Reasoning in Terminological Knowledge Representation Systems*

Martin Buchheit†
Deutsches Forschungszentrum für KI (DFKI)
Stuhlsatzenhausweg 3
D-6600 Saarbrücken, Germany
e-mail: buchheit@dfki.uni-sb.de

Francesco M. Donini, Andrea Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
e-mail: {donini,aschaerf}@assi.ing.uniroma1.it

Abstract

Terminological Knowledge Representation Systems (TKRS) are tools for designing and using knowledge bases that make use of terminological languages (or concept languages). The TKRS we consider in this paper is of practical interest since it goes beyond the capabilities of presently available TKRS. First, our terminological language is highly expressive. In particular, it is equipped with a concept language, called $\mathcal{ALCNR}$, including general complements of concepts, number restrictions and role conjunction. Second, it allows one to express inclusion statements between general concepts, which means in particular to express terminological cycles. We provide a sound, complete and terminating calculus for reasoning in $\mathcal{ALCNR}$-knowledge bases based on the general technique of constraint systems.

1 Introduction

A general characteristic of many proposed Terminological Knowledge Representation Systems (TKRS) such as BACK, LOOM, CLASSIC, $\mathcal{KRIS}$, [Ric91, WS92] is that they are made up of two different components. Informally speaking, the first is a general schema concerning the classes of individuals to be represented, their general properties and mutual relationships, while the second is a (partial) instantiation of this schema, containing assertions relating either individuals to classes, or individuals to each other.

Retrieving information in actual knowledge bases (KBs) built up using one of these systems is a deductive process involving both the schema (TBox) and its instantiation (ABox).

During the realization and use of a KB, a TKRS should provide a mechanical solution for at least the following problems (from now on, we use the word *concepts* to refer to classes):

*This work has been supported by the ESPRIT Basic Research Action N.6810 COMPULOG 2 and by the Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of the CNR (Italian Research Council).

†This research was partly done while the first author was visiting the Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

1. *Concept Satisfiability*: given a KB and a concept C, does there exist at least one model of the KB assigning a nonempty extension to C?

2. *Subsumption*: given a KB and two concepts C and D, is C more general than D in any model of the KB?

3. *KB-satisfiability*: are an ABox and a TBox consistent with each other?

4. *Instance Checking*: given a KB, an individual a and a concept C, is a an instance of C in any model of the KB?

Up to now, all the proposed systems (except for $\mathcal{KRIS}$) give incomplete procedures for solving the above problems 1–4. That is, some inferences are missed, in some cases without a precise semantical characterization of which ones are. If the designer or the user needs a (more) complete reasoning, she/he must either write programs in a suitable programming language, or define appropriate inference rules completing the inference capabilities of the system (as in BACK, LOOM, and CLASSIC).

Nevertheless, in our opinion incomplete procedures are just a provisional answer to the problem—the best possible up to now. In order to improve on such an answer, a theoretical analysis of the general problems 1–4 must be done. But most importantly, theoretical analysis is needed for making cyclic definitions of concepts (see [Neb90, Chapter 5]) fully available in TKRS. Such a feature is of undoubtable practical interest, yet present TKRS can only approximate cycles, by using forward inference rules.

Previous results do not deal with the problems 1–4 in their full generality. The problems are studied in [Neb90, Chapter 4], but giving only incomplete procedures, and not dealing with cycles. In [DLNS92] complexity of instance checking has been analyzed, but considering only KBs without a TBox. Previous theoretical work on cycles was done in [Baa90b, Baa90a, BBH+90, Neb90, Neb91, Sch91], but considering KBs formed by the TBox alone. Moreover, these approaches do not deal with number restrictions (except for [Neb90, Section 5.3.5]), which are a basic feature already provided by TKRS, and the techniques used seem not easily extensible to reasoning with ABoxes.

In this paper, we propose a TKRS equipped with a highly expressive language of practical significance, and prove decidability of problems 1–4. In particular, our system makes use of the language $\mathcal{ALCNR}$, which supports general complements of concepts, number restrictions and role conjunction. Moreover, the system allows one to express inclusion statements between general concepts and, as a particular case, terminological cycles. We prove decidability by means of a suitable calculus, which is developed within the quite well established framework of constraint systems (see [DLNN91, SSS91, DLNS91]) thus exploiting a uniform approach to reasoning in TKRS. Moreover, our calculus can be easily turned into a decision procedure.

The paper is organized as follows. In Section 2 we introduce the language, and we give it a Tarski-style extensional semantics, which is the most commonly used. In Section 3 we provide a calculus, and show its correctness and termination. In Section 4 we consider a refinement of our calculus, working in exponential space. In Section 5 we compare our approach with previous results on decidable TKRS, and we establish the equivalence of general inclusion statements and general concept definitions using the descriptive semantics. Finally, we discuss in detail several practical impacts of our results in Section 6. For the sake of brevity proofs are omitted. They can be found in [BDS93].

2 Preliminaries

In concept languages, concepts represent the classes of objects in the domain of interest, while roles represent binary relations between objects. Complex concepts and roles can be defined by means of suitable constructors applied to primitive concepts and primitive roles. In particular, concepts and roles in $\mathcal{ALCNR}$ can be formed by means of the following syntax (A denotes a primitive concept, P_i $(i = 1, \ldots, m)$ denotes a primitive role, C and D denote arbitrary concepts and R an arbitrary role):

$$
\begin{array}{rcll}
C, D & \rightarrow & A \mid & \text{(primitive concept)} \\
 & & \top \mid & \text{(top)} \\
 & & \bot \mid & \text{(bottom)} \\
 & & (C \sqcap D) \mid & \text{(intersection)} \\
 & & (C \sqcup D) \mid & \text{(union)} \\
 & & \neg C \mid & \text{(complement)} \\
 & & \forall R.C \mid & \text{(universal quantification)} \\
 & & \exists R.C \mid & \text{(existential quantification)} \\
 & & (\geq n\, R) \mid (\leq n\, R) & \text{(number restrictions)} \\
R & \rightarrow & P_1 \sqcap \cdots \sqcap P_m & \text{(role conjunction)}
\end{array}
$$

We interpret concepts as subsets of a domain and roles as binary relations over a domain. More precisely, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *extension function* of $\mathcal{I}$) that maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that the following equations are satisfied ($\sharp\{\}$ denotes the cardinality of a set):

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow\ d_2 \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\} \\
(\geq n\, R)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid\ \ \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\}\ \ \geq n\} \\
(\leq n\, R)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid\ \ \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\}\ \ \leq n\} \\
(P_1 \sqcap \cdots \sqcap P_m)^{\mathcal{I}} &= P_1^{\mathcal{I}} \cap \cdots \cap P_m^{\mathcal{I}}
\end{aligned}
$$

A KB built by means of concept languages is formed by two components: The *intensional* one, called TBox, and the *extensional* one, called ABox.

We first turn our attention to the TBox. As we said before, the intensional level specifies the properties of the concepts of interest in a particular application. Syntactically, such properties are expressed in terms of so-called *inclusion statements* (see [Neb90, Chapter 3]). An inclusion statement (or simply inclusion) has the form

$$C \sqsubseteq D$$

where C and D are two arbitrary concepts. Intuitively, the statement specifies that every instance of C is also an instance of D. More precisely, an interpretation $\mathcal{I}$ *satisfies* the inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A TBox $\mathcal{T}$ is a finite set of inclusions. An interpretation $\mathcal{I}$ is a *model* for $\mathcal{T}$ if $\mathcal{I}$ satisfies all inclusions in $\mathcal{T}$.

In comparison to many TKRS (e.g. [Neb90]) our TBox statements are more expressive in the following sense. Those systems usually provide the user with mechanisms for *concept definitions* of the form $A \sqsubseteq D$ (*inclusion*), or $A \doteq D$ (*equivalence*), with the restrictions that the left-hand side concept A must be a primitive concept (i.e. a concept name), that for each primitive concept at most one statement is allowed, and that no terminological cycles are allowed, i.e. no primitive concept may occur—neither directly nor indirectly—within its own definition.

We do not impose any of these restrictions to the form of inclusions, obtaining statements that are strictly more expressive than the ones described above. In particular, an equivalence of the form $A \doteq D$ can be expressed in our system using the pair of inclusions $A \sqsubseteq D$ and $D \sqsubseteq A$, whereas an inclusion of the form $C \sqsubseteq D$, where C and D are arbitrary concepts, cannot be expressed with the restrictions imposed by other systems. Moreover, cyclic inclusions are allowed in our statements, realizing the so-called *terminological cycles*.

As shown in [Neb91], there are at least three types of semantics for terminological cycles inside concept definitions, namely the least fixed point, the greatest fixed point, and the descriptive semantics. However, fixed point semantics apply only to fixed point statements, which are much less general than our inclusion statements. Instead, the descriptive semantics interprets statements as just restricting the set of possible models, with no definitional import. Hence, it can be suitably extended to our case, and is exactly the one we adopt.

We can now turn our attention to the *extensional level* of the KB, i.e. the ABox. The ABox essentially allows one to specify instance-of relations between individuals and concepts, and between pairs of individuals and roles.

Let $\mathcal{O}$ be an alphabet of symbols, called *individuals*. Instance-of relationships are expressed in terms of *membership assertions* of the form:

$$C(a), \qquad R(a,b)$$

where a and b are individuals, C is a concept, and R is a role. Intuitively, the first form states that a is an instance of C, whereas the second form states that a is related to b by means of the role R.

In order to assign a meaning to membership assertions, the extension function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I}$ is extended to individuals by mapping them to elements of $\Delta^{\mathcal{I}}$ in such a way that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (Unique Name Assumption). An interpretation $\mathcal{I}$ *satisfies* the assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and *satisfies* $R(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An ABox $\mathcal{A}$ is a finite set of membership assertions. $\mathcal{I}$ is a *model* for an ABox $\mathcal{A}$ if $\mathcal{I}$ satisfies all the assertions in $\mathcal{A}$.

An $\mathcal{ALCNR}$*-knowledge base* Σ is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox, and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is a *model* for Σ if it is both a model for $\mathcal{T}$ and a model for $\mathcal{A}$. We can now formally define the problems 1–4 mentioned in the introduction, w.r.t. a given KB Σ:

1. *Concept Satisfiability*: C is *satisfiable* w.r.t Σ, if there exists a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$;

2. *Subsumption*: C is *subsumed* by D w.r.t. Σ, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of Σ;

3. *KB-satisfiability*: Σ is *satisfiable* if it has a model;

4. *Instance Checking*: a is an instance of C, written $\Sigma \models C(a)$, if the assertion $C(a)$ is satisfied in every model of Σ.

Example 2.1 Consider the following KB $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$:

$$\mathcal{T} = \{\ \exists \texttt{TEACHES.Course} \sqsubseteq (\texttt{Student} \sqcap \exists \texttt{DEGREE.BS}) \sqcup \texttt{Prof},$$
$$\texttt{Prof} \sqsubseteq \exists \texttt{DEGREE.MS},\ \exists \texttt{DEGREE.MS} \sqsubseteq \exists \texttt{DEGREE.BS},$$
$$\texttt{MS} \sqcap \texttt{BS} \sqsubseteq \bot\}$$
$$\mathcal{A} = \{\texttt{TEACHES(john, cs1)},\ (\leq 1\ \texttt{DEGREE})\texttt{(john)},\ \texttt{Course(cs1)}\}$$

It is easy to see that Σ is satisfiable. Notice also that it is possible to draw several non-trivial conclusions from Σ. For example, we can infer that $\Sigma \models \texttt{Student(john)}$. Intuitively this can be shown as follows: john teaches a course, then he is either a student with a BS or a professor. But he can't be a professor since professors have at least two degrees (BS and MS) and he has at most one, therefore he is a student. □

Given the above semantics, the problems 1–4 can all be reduced to KB-satisfiability (or to its complement) in linear time. In fact, given a KB $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, a concept C is satisfiable w.r.t Σ iff the KB $\langle \mathcal{T}, \mathcal{A} \cup \{C(b)\} \rangle$ is satisfiable, and C is subsumed by D w.r.t. Σ iff the KB $\langle \mathcal{T}, \mathcal{A} \cup \{(C \sqcap \neg D)(b)\} \rangle$ is not satisfiable, where b is a new individual not appearing in Σ. Finally, $\Sigma \models C(a)$ iff the KB $\langle \mathcal{T}, \mathcal{A} \cup \{(\neg C)(a)\} \rangle$ is not satisfiable. Hence, we can concentrate just on KB-satisfiability in the next section.

3 Decidability Result

In this section we present a calculus for deciding KB-satisfiability and state its correctness and termination.

Our method makes use of the notion of constraint system [DLNN91, SSS91], and is based on a tableaux-like calculus [Fit90] that tries to build a model for the logical formula corresponding to a KB.

Consider an alphabet of variable symbols $\mathcal{V}$. The elements of $\mathcal{V}$ are denoted by the letters x, y, z, w. In the sequel we use the term *object* as an abstraction for individual and variable (i.e. an object is an element of $\mathcal{O} \cup \mathcal{V}$). Objects are denoted by the symbols s, t and, as in Section 2, individuals are denoted by a, b. In the rest of this section, R denotes the role $R = P_1 \sqcap \ldots \sqcap P_k$ $(k \geq 1)$.

A *constraint* is a syntactic entity of one of the forms:

$$s{:}\,C, \quad sPt, \quad \forall x.x{:}\,C, \quad s \not\doteq t,$$

where C is a concept and P is a primitive role. Concepts are assumed to be *simple*, i.e. they contain only complements of the form $\neg A$, where A is a primitive concept. Arbitrary $\mathcal{ALCNR}$-concepts can be rewritten into equivalent simple concepts in linear time [DLNN91]. A *constraint system* is a finite nonempty set of constraints.

Given an interpretation $\mathcal{I}$, we define an *$\mathcal{I}$-assignment* α as a function that maps every variable in $\mathcal{V}$ to an element of $\Delta^{\mathcal{I}}$ (not necessarily injectively), and every individual a to $a^{\mathcal{I}}$ (i.e. $\alpha(a) = a^{\mathcal{I}}$ for all $a \in \mathcal{O}$).

A pair $(\mathcal{I}, \alpha)$ *satisfies* the constraint $s{:}\,C$ if $\alpha(s) \in C^{\mathcal{I}}$, the constraint sPt if $(\alpha(s), \alpha(t)) \in P^{\mathcal{I}}$, the constraint $s \not\doteq t$ if $\alpha(s) \neq \alpha(t)$, and finally, the constraint $\forall x.x{:}\,C$ if $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$. A constraint system S is *satisfiable* if there is a pair $(\mathcal{I}, \alpha)$ that satisfies every constraint in S.

An $\mathcal{ALCNR}$-knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ can be translated into a constraint system S_Σ by replacing every inclusion $C \sqsubseteq D \in \mathcal{T}$ with the constraint $\forall x.x{:}\,\neg C \sqcup D$, every

membership assertion $C(a)$ with $a{:}\,C$, $R(a,b)$ with $aP_1b,\ldots,aP_kb$, and including the constraint $a \not\doteq b$ for every pair (a,b) of individuals appearing in $\mathcal{A}$. It is easy to see that Σ is satisfiable if and only if S_Σ is satisfiable.

In order to check a constraint system S for satisfiability, our technique adds constraints to S until either an evident contradiction is generated or an interpretation satisfying it can be obtained from the resulting system. Constraints are added on the basis of a suitable set of so-called *propagation rules.*

Before providing the rules, we need some additional definitions. Let S be a constraint system. We say that t is an *R-successor of s in S* if $sP_1t,\ldots,sP_kt$ are in S. We say that t is a *1-successor of s in S* if for some role R, t is an R-successor of s. If S is clear from the context we simply say that t is an R-successor or a 1-successor of s. Moreover, we denote by *successor* the transitive closure of the relation 1-successor, and we denote by *predecessor* its inverse.

We denote by $S[x/s]$ the constraint system obtained from S by replacing each occurrence of the variable x by s. We say that s and t are *separated in S* if the constraint $s \not\doteq t$ is in S.

Given a constraint system S we say that two variables x and y are *S-equivalent*, written $x \equiv_s y$, if $\{C \mid x{:}\,C \in S\} = \{C \mid y{:}\,C \in S\}$. Intuitively, two S-equivalent variables could represent the same element in the potential interpretation built by the rules, unless they were separated.

The *propagation rules* are:

$S \rightarrow_\sqcap \{s{:}\,C_1,\ s{:}\,C_2\} \cup S$

if $s{:}\,C_1 \sqcap C_2$ is in S, $s{:}\,C_1$ and $s{:}\,C_2$ are not both in S

$S \rightarrow_\sqcup \{s{:}\,D\} \cup S$

if $s{:}\,C_1 \sqcup C_2$ is in S, neither $s{:}\,C_1$ nor $s{:}\,C_2$ is in S and $D = C_1$ or $D = C_2$

$S \rightarrow_\forall \{t{:}\,C\} \cup S$

if $s{:}\,\forall R.C$ is in S, t is a R-successor of s, $t{:}\,C$ is not in S

$S \rightarrow_\exists \{sP_1y,\ldots,sP_ky,\ y{:}\,C\} \cup S$

if $s{:}\,\exists R.C$ is in S, y is a new variable, there is no t such that t is a R-successor of s in S and $t{:}\,C$ is in S; if s is a variable there is no variable w in S such that w is a predecessor of s and $s \equiv_s w$

$S \rightarrow_\geq \{sP_1y_i,\ldots,sP_ky_i \mid i \in 1..n\} \cup$
$\{y_i \not\doteq y_j, \mid i,j \in 1..n, i \neq j\} \cup S$

if $s{:}\,(\geq n\,R)$ is in S, $y_1,\ldots,y_n$ are new variables, there do not exist n pairwise separated R-successors of s in S; if s is a variable there is no variable w such that w is a predecessor of s and $s \equiv_s w$

$S \rightarrow_\leq S[y/t]$

if $s{:}\,(\leq n\,R)$ is in S, s has more than n R-successors in S, y,t are two R-successors of s that are not separated

$S \rightarrow_{\forall x} \{s{:}\,C\} \cup S$

if $\forall x.x{:}\,C$ is in S, s appears in S, $s{:}\,C$ is not in S.

We call the rules $\rightarrow_{\sqcup}$, $\rightarrow_{\leq}$ *nondeterministic* rules, since they can be applied in different ways to the same constraint system. All the other rules are called *deterministic* rules. Moreover, we call the rules $\rightarrow_{\exists}$, $\rightarrow_{\geq}$ *generating* rules, since they introduce new variables in the constraint system. All other rules are called *nongenerating* ones.

The use of the condition based on the S-equivalence relation in the generating rules is related to the goal of keeping the constraint system finite even in presence of potentially infinite chains of applications of generating rules. Its role will become clearer in the sequel.

One can verify that rules are always applied to a system S either because of the presence in S of a given constraint $s\!:\!C$ or, in the case of the $\rightarrow_{\forall x}$-rule, because of the presence of an object s in S. When no confusion arises, we will say that a rule is *applied to* the object s.

Proposition 3.1 (Invariance) *Let S and S' be constraint systems. Then:*

1. *If S' is obtained from S by application of a deterministic rule, then S is satisfiable if and only if S' is satisfiable.*
2. *If S' is obtained from S by application of a nondeterministic rule, then S is satisfiable if S' is satisfiable. Furthermore, if a nondeterministic rule applies to S, then it can be applied in such a way that it yields a constraint system S' which is satisfiable if and only if S is satisfiable.*

Given a constraint system, more than one rule might be applicable to it. We define the following *strategy* for the application of rules:

1. apply a rule to a variable only if no rule is applicable to individuals;
2. apply generating rules only if no nongenerating rule is applicable;
3. apply a generating rule to a variable x only if no rule is applicable to a predecessor of x.

A constraint system is *complete* if no propagation rule applies to it. A complete system derived from a system S is also called a *completion* of S. A *clash* is a constraint system having one of the following forms:

- $\{s\!:\!\bot\}$
- $\{s\!:\!A,\ s\!:\!\neg A\}$, where A is a primitive concept.
- $\{s\!:\!(\leq n\, R)\} \cup \{sP_1t_i, \ldots, sP_kt_i \mid i \in 1..n+1\}$
  $\cup \{t_i \neq t_j \mid i,j \in 1..n+1, i \neq j\}$.

In order to obtain an interpretation from a complete constraint system we need some additional definitions. Let S be a constraint system and x, w be variables in S. We call w a *witness of x in S* if the three following conditions hold:

1. $x \equiv_s w$
2. w is a predecessor of x in S
3. no rule is applicable to any predecessor of x.

Notice that the third condition ensures that no new constraint will be imposed on x. We say x *is blocked* (*by* w) if x has a witness (w) in S. In a constraint system obtained from an S_Σ by applying the rules according to the strategy, a blocked variable has exactly one witness and no successors.

Let S be a constraint system. We define the *canonical interpretation* $\mathcal{I}_S$ for S and the *canonical* $\mathcal{I}_S$*-assignment* α_S for S as follows:

1. $\Delta^{\mathcal{I}_S} := \{s \mid s \text{ is an object in } S\}$
2. $\alpha_S(s) := s$
3. $s \in A^{\mathcal{I}_S}$ iff $s\colon A$ is in S
4. $(s,t) \in P^{\mathcal{I}_S}$ iff
   (a) sPt is in S or
   (b) s is a blocked variable, w is the witness of s in S and wPt is in S.

The canonical interpretation of complex concepts and roles can be uniquely reconstructed from the interpretations of the primitive ones imposing the equations stated in Section 2.

Theorem 3.2 (Correctness) *Let S be a complete constraint system. S is satisfiable iff it contains no clash.*

Proof. (Sketch) If S contains a clash, it is clearly unsatisfiable. If S contains no clash, it can be shown that the pair $(\mathcal{I}_S, \alpha_S)$ satisfies every constraint in S. □

The termination of the calculus is based on the following property: In any constraint system obtained from an S_Σ by applying the rules according to the strategy, every variable can have at most 2^n variables among its predecessors, where n is the size of Σ.

Theorem 3.3 (Termination) *Let S be a constraint system. Every completion of S is finite.*

We come now to the main result of this section.

Theorem 3.4 (Decidability) *Checking whether an $\mathcal{ALCNR}$-KB is satisfiable is a decidable problem.*

Notice that, since the domain of the canonical interpretation $\Delta^{\mathcal{I}_S}$ is always finite, we have also implicitly proved that $\mathcal{ALCNR}$-knowledge bases have the *finite model property*, i.e. any satisfiable knowledge base has a finite model. This property has been extensively studied in modal logics [HC84] and dynamic logics [Har84]. In particular a technique, called *filtration*, has been developed both to prove the finite model property and to build a finite model for a satisfiable formula. This technique allows one to build a finite model from an infinite one by grouping the worlds of a structure in equivalence classes based on the set of formulae that are satisfied in each world. It is interesting to observe that our calculus, based on witnesses, can be considered as a variant of the filtration technique where the equivalence classes are determined on the basis of our S-equivalence relation. However, because of number restrictions, variables that are S-equivalent cannot be grouped, since they might be separated (e.g. they might have been introduced by the same application of the $\rightarrow_{\geq}$-rule). Nevertheless, they can have the same successors, as stated in point 4.(b) of the definition of canonical interpretation. This would correspond to grouping variables of an infinite model in such a way that separations are preserved.

4 A Calculus Working in Exponential Space

The calculus proposed in the previous section requires to compute all the completions of the constraint system S_Σ. Unfortunately, such completions may be of double exponential size w.r.t. the size of Σ.

For an exponential space algorithm it is therefore crucial not to keep an entire complete constraint system in memory, but to store only small portions at a time.

We give propagation rules, called *trace rules*, that build up only a portion of complete constraint systems.

The *trace rules* consist of the $\rightarrow_{\sqcap}$-, $\rightarrow_{\sqcup}$-, $\rightarrow_{\forall}$-, $\rightarrow_{\forall x}$- and the $\rightarrow_{\leq}$-rule (the nongenerating rules) together with the two generating rules $\rightarrow_{T\exists}$ and $\rightarrow_{T\geq}$, which are obtained respectively from the $\rightarrow_{\exists}$- and the $\rightarrow_{\geq}$-rule adding the following condition of application:

$$\text{for all constraints } tPx \text{ in } S,\ t \text{ is a predecessor of } s \text{ or } t = s.$$

Let T be a constraint system obtained from S_Σ by application of the trace rules. We call T a *trace* of S_Σ if no trace rule applies to T.

If the trace rules are applied according to the strategy they show the following behavior: Given an object s, if at least one generating rule is applicable, all its 1-successors $y_1, \ldots, y_n$ are introduced. Then, after nongenerating rules are applied, one variable y_i is (nondeterministically) chosen, and all 1-successors of y_i are introduced. Differently from normal propagation rules, no successor is introduced for any object different from y_i. Then, one variable is chosen among the 1-successors of y_i, only its 1-successors are added to the constraint system, and so on.

The reason why we introduce all the 1-successors of the "chosen" object is the following. For every chosen object s all 1-successors of s must be present simultaneously at some stage of the computation, since only the interplay of role intersections and number restrictions forces us to identify certain successors. This is important because, when identifying variables, the constraints imposed on them are combined, which may lead to clashes that otherwise would not have occurred.

A calculus based on trace rules, for deciding satisfiability and subsumption of $\mathcal{ALCNR}$-concepts, was previously defined in [DLNN91]. Algorithms exploiting traces have been given in [HNSS90]. However, all these works do not deal with KB, but only with concept expressions.

Proposition 4.1 *Let Σ be an $\mathcal{ALCNR}$-KB, S_Σ its associated constraint system, and let n be the size of Σ. Then:*

1. *The length of a trace rule derivation issuing from S_Σ is bounded by 2^n.*
2. *Every complete constraint system extending S_Σ can be obtained as the union of finitely many traces.*
3. *Suppose S is a complete constraint system extending S_Σ and $\mathcal{T}$ is a finite set of traces such that $S' = \bigcup_{T \in \mathcal{T}} T$. Then S' contains a clash if and only if some $T \in \mathcal{T}$ contains a clash.*

Since an $\mathcal{ALCNR}$-KB Σ is satisfiable if and only if there exists a complete constraint system derivable from S_Σ without a clash, it follows from Proposition 4.1 that satisfiability of an $\mathcal{ALCNR}$-KB can be decided with exponential space. A possible algorithm using space $2^{p(n)}$ where $p(n)$ is a polynomial in the size of S_Σ may be the following: compute one complete constraint system, one trace at a time, trying all possible applications of the nondeterministic rules $\rightarrow_{\leq}$, $\rightarrow_{\sqcup}$, and finding the choices leading to traces without a clash.

Theorem 4.2 *Satisfiability of an $\mathcal{ALCNR}$-KB can be decided with exponential space.*

Thanks to an unpublished manuscript by D. McAllester, and (independently) from an observation by W. Nutt, we know that deciding the satisfiability of an $\mathcal{ALCNR}$-KB is an EXPTIME-hard problem. Hence, we do not expect to find any algorithm solving the problem in polynomial space, unless PSPACE=EXPTIME. Nevertheless, the algorithm outlined above may require double exponential time. It is still open whether there exists an algorithm working in (simple) exponential time.

5 Relation to previous work

There have been several works about reasoning with inclusions and terminological cycles and several different approches and reasoning techniques have been proposed.

In [Baa90b] and [Neb91] a characterization of the subsumption problem is given with the help of automata, whereas the technique exploited in [BBH+90] and [Baa90a] is based on the notion of *concept tree*. The results in [Sch91] are obtained by establishing a correspondence between concept languages and Propositional Dynamic Logics (PDL), and reducing the given problem to a satisfiability problem in PDL.

However, all these approaches, i.e. reduction to automata problems, concept trees and reduction to PDL, deal only with TBoxes and they don't seem to be suitable to deal also with ABoxes. On the other hand, the constraint system technique, even though it was conceived for TBox-reasoning, can be easily extended to ABox-reasoning, as also shown in [Hol90],[BH91] and [DLNS92].

Now we compare the expressive power of TBoxes defined as a set of inclusions (as done in this paper) and TBoxes defined as a set of (possibly cyclic) concept definitions of the form $A \sqsubseteq D$ and $A \doteq D$.

Unlike [Baa90a] and [Sch91], we consider reasoning problems dealing with TBox and ABox together. Moreover, we use the descriptive semantics for the concept definitions, as we do for the inclusions. The result we have obtained is that inclusion statements and concept definitions have actually the same expressive power. In details, we show that the satisfiability of a knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$, where $\mathcal{T}$ is a set of inclusion statements can be reduced to the satisfiability of a knowledge base $\Sigma' = \langle \mathcal{A}', \mathcal{T}' \rangle$ such that $\mathcal{T}'$ is a set of concept definitions. The other direction—from concept definitions to inclusions—is trivial since definitions can be expressed by pairs of inclusions, as already mentioned in Section 2.

As a notation, given a TBox $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$, we define the concept $C_{\mathcal{T}}$ as $C_{\mathcal{T}} = (\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$. As pointed out in [Baa90a] for $\mathcal{ALC}$, an interpretation satisfies a TBox $\mathcal{T}$ iff it satisfies the equation $C_{\mathcal{T}} = \top$. This result easily extends to $\mathcal{ALCNR}$.

Given a knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ and a concept A not appearing in Σ, we define the knowledge base $\Sigma' = \langle \mathcal{A}', \mathcal{T}' \rangle$ as follows:

$$\begin{aligned} \mathcal{A}' &= \mathcal{A} \cup \{A(b) \mid b \text{ is an individual in } \Sigma\} \\ \mathcal{T}' &= \{A \sqsubseteq C_{\mathcal{T}} \sqcap \forall P_1.A \cdots \sqcap \forall P_n.A\} \end{aligned}$$

where $P_1, P_2, \ldots, P_n$ are all the primitive roles appearing in Σ.

Theorem 5.1 *Σ is satisfiable iff Σ' is satisfiable.*

6 Conclusion

In this paper we have proved the decidability of the main inference services of a TKRS based on the concept language $\mathcal{ALCNR}$. We believe that this result is not only of theoretical importance, but has the following impacts on existing TKRS.

First of all, a complete procedure working in exponential space can be easily devised from the calculus provided in Sections 3 and 4. From this procedure, one can build more efficient (but still complete) ones by applying optimization techniques. Such procedure might work well in practical cases, despite of its worst case intractability.

Secondly, a complete procedure (possibly optimized) offers a benchmark for comparing incomplete procedures, not only in terms of performance, but also in terms of missed inferences. In fact, incomplete procedures can be meaningfully compared only if missed inferences are considered. However, to recognize missed inferences over large examples, one needs exactly a complete procedure—even if not an efficient one—like ours.

Thirdly, new incomplete procedures can be obtained from the calculus by modifying some of the propagation rules. Since the rules build up a model, modifications to them have a semantical counterpart which gives a precise account of the incomplete procedures obtained. For instance, define the depth of a variable x as the number of variables which are predecessors of x. Then, an incomplete calculus could be devised, which generates variables only to a given depth—say, linear depth in the size of the KB. This calculus would miss contradictions (and hence inferences, by refutation) occurring in variables which are "far away" from the known individuals of the KB, and this is a meaningful explanation of the incompleteness, even for a non-expert user. From a computational point of view, an immediate consequence of the complexity analysis carried over in this paper is that such an incomplete procedure would run in polynomial space.

Acknowledgements

We would like to thank Maurizio Lenzerini for several discussions that contributed to the paper, Franz Baader and Werner Nutt for many helpful comments on earlier drafts. The third author also acknowledges Yoav Shoham for his hospitality at the Computer Science Department of the Stanford University.

References

[Baa90a] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Technical Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1990. A shorter version appeared in *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, pages 446–451, 1991.

[Baa90b] F. Baader. Terminological cycles in KL-ONE-based KR-languages. In *Proc. of the 8th Nat. Conf. on Artificial Intelligence AAAI-90*, pages 621–626, 1990.

[BBH+90] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, and J. H. Siekmann. Concept logics. In J.W. Lloyd, editor, *Computational Logics, Symposium Proceedings*, pages 177–201. Springer Verlag, 1990.

[BDS93] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. Technical Report RR-93-10, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, 1993.

[BH91] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence. Springer Verlag, 1991.

[DLNN91] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91*, pages 151–162. Morgan Kaufmann, 1991.

[DLNS91] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. A hybrid system integrating datalog and concept languages. In *Proc. of the 2nd Italian Conf. on Artificial Intelligence*, number 549 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1991.

[DLNS92] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. From subsumption to instance checking. Technical Report 15.92, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1992.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Springer Verlag, New York, 1990.

[Har84] David Harel. Dynamic logic. In *Handbook of Philosophical Logic*, volume 2, pages 497–640. D. Reidel, Dordrecht, Holland, 1984.

[HC84] George E. Hughes and M. J. Cresswell. *A Companion to Modal Logic.* Methuen, London, 1984.

[HNSS90] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proc. of 9th the European Conf. on Artificial Intelligence ECAI-90*, pages 348–353, London, 1990. Pitman.

[Hol90] Bernhard Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. In *German Workshop on Artificial Intelligence.* Springer Verlag, 1990.

[Neb90] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Number 422 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1990.

[Neb91] B. Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, 1991.

[Ric91] C. Rich, editor. SIGART bulletin. Special issue on implemented knowledge representation and reasoning systems. (2)3, June 1991.

[Sch91] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, Sydney, 1991.

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[WS92] W.A. Woods and J.G. Schmolze. The KL-ONE family. In F.W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992.

The Complexity of Proof-Search with Analytic Tableaux and Related Systems. Extended Abstract

Marcello D'Agostino
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ (England)
mda@uk.ac.ic.doc

Marco Mondadori
Istituto di Discipline Filosofiche
Università di Ferrara
via Savonarola 38, 44100 Ferrara (Italy)
G9QFEVE1@ICINECA.EARN

28 November 1992

1 Introduction

The aim of this paper is twofold. Firstly, we compare the complexity of proof-search algorithms based on the system of analytic tableaux with that of algorithms based on the related system **KE** [Mon88]. We have shown elsewhere [D'A90] that **KE** linearly simulates the tableau method but the tableau method cannot p-simulate **KE** and that, in fact, analytic tableaux, in their standard formulation, are affected by several anomalies which culminate in the surprising fact [D'A92] that they cannot p-simulate the truth-tables. However, many researchers argue that results expressed in terms of polynomial simulation refer to the relative length of minimal proofs in different systems but do not say much about the relative difficulty of *proof-search*. Here we show how the methodology of p-simulation can be developed in order to overcome this criticism, and show that the (analytic) **KE** system is an improvement of analytic tableaux, in a particularly strong sense, also from the point of view of proof-search. Next we address the problem of coping with the exponential worst-case complexity of proof-search in the analytic **KE** system which, by a result of Ajtai [Ajt88], extends to all analytic propositional systems and (by the well-known conjecture that $P \neq NP$) probably to all propositional systems *tout court*. It is not difficult to identify in the only branching rule of the **KE** system — namely the analytic cut rule PB — the origin of combinatorial explosion. So an obvious strategy to cope with this problem consists in imposing an upper bound on the number of allowed applications of this rule. It turns out that all such resource-bounded **KE** systems have a polynomial time decision procedure (the degree of the polynomial depending on the given bound) and the fragments of classical propositional logic covered by resource-bounded **KE** systems of low-degree (even by the simplest system which allows for *no* application of PB) are quite substantial and may suffice for many applications. Notice that a similar strategy would lead nowhere if applied to the standard tableau method, since the systems obtained by bounding the number of branchings would be very weak. We also examine to what extent it is possible to provide an indipendent characterization of the non-classical logics for which the resource-bounded systems are complete.

2 The relative complexity of proof-search

In [Bib82], Wolfgang Bibel wrote: "[...] evaluation of the performance of theorem provers is quite a complicated thing, so that it would be helpful to have a clearer view of what we mean by 'quantitatively better' or by 'improved methods'. In the former case one would compare the relative performance of implementations of different methods on a number of samples. Since [...] there is

relatively little experience in the current state of the art of building and testing theorem provers, any such experimental comparisons at present should be taken with much caution. [...] Under these circumstances it is not surprising that the present techniques of mathematical analysis, the other possibility of measuring performance, are rather limited as well. For such an analysis, what we would need is a realistic mathematical model of the binary relation which captures the natural and practically relevant meaning of the term 'better than' with respect to proof procedures." The situation is even more complicated, if we consider that what we often intend to compare are non-deterministic *proof systems* rather than deterministic *proof procedures*. A proof systems is a collection of inference rules with only partial or no control on their application, so that in some cases we choose which rule to apply next among several possibilities. Rational decisions on the relative computational merits of proof systems are important when selecting a system as the underlying formalization of future algorithmic developments. From this point of view, a comparison between deterministic algorithms based on different formal systems (namely on two different sets of allowed inference rules) is not a crucial test unless the algorithms in question can be proved to be *optimal*. Otherwise negative empirical evidence can always be blamed on the particular "current version" of the algorithm and diverted from the formalization itself. However, there is a natural limit on the possible algorithms which can be developed on the basis of a given formalization. The aim of this section is to address the problem of building a model of the notion "the proof systems S_2 is an improvement of the proof systems S_1" which is relevant to the complexity of *proof-search* within different systems. As a case-study we shall analyse the relative complexity of proof-search in analytic tableaux and in the system **KE** [Mon88, D'A90].

We start by clarifying the related notions of *proof system*, *polynomial simulation* and proof procedure.

Definition 1 We define a *proof system* as a triple (Σ, R, f) where Σ is a finite alphabet, R is a relation $\Sigma^* \times \Sigma^*$ computable in polynomial time and f is a partial function $C_R \mapsto 2^{\mathcal{F}}$ computable in polynomial time, with C_R the closure of the empty string Λ under the relation R, and $\mathcal{F}$ the subset of Σ^* corresponding to the logical formulas.

The alphabet Σ includes all the usual logical symbols plus symbols specific to the proof system. The relation R defines the notion "the proof-configuration T_2 is an elementary extension of the proof-configuration T_1" and so the closure C_R of the empty configuration under R generates the set of all possible proof-configurations. (In general, a proof configuration may be something more than a single derivation, i.e. a connected graph of formulas: from the viewpoint of proof search it may be, in some cases, more convenient to define a proof configuration as a, possibly structured, set of such derivations. For instance, in a natural deduction system, it may be a tree of proof-trees.) The function f identifies the formula(s) proved by a proof-configuration (it may be undefined for some proof-configurations). The requirement that both R and f are polynomial time computable amounts to the requirement that the proof systems is a "sensible" one (both an elementary step in a proof and recognizing the formula(s) proved by a proof are computationally easy tasks).

Definition 2 Given proof systems $S_1 = (\Sigma_1, R_1, f_1)$ and $S_2 = (\Sigma_2, R_2, f_2)$, a *polynomial simulation* of S_1 in S_2 is a function $s : C_{R_1} \mapsto C_{R_2}$ computable in polynomial time and such that for every $\pi \in C_{R_1}$, $f_1(\pi) = f_2(s(\pi))$.

So, a polynomial simulation maps, in polynomial time, proof-configurations of one systems into equivalent proof-configurations of the other.

Definition 3 A *proof procedure* is a quadruple (S, g, F, P), where $S = (\Sigma, R, f)$ is a proof system, g is a function from logical formulas to proof-configurations called the *initializing function*, F is a polynomial time functional restriction of R, and P is a function $\mathcal{F} \times C_R \mapsto C_R$ defined as follows (recall Λ is the empty configuration and we denote by $x : \pi$ a non-empty configuration starting with the symbol x):

$$
\begin{aligned}
P(A, \Lambda) &= P(A, g(A)) \\
P(A, x : \pi) &= \begin{cases} x : \pi & \text{if } A \in f(x : \pi) \\ P(A, F(x : \pi)) & \text{otherwise} \end{cases}
\end{aligned}
$$

Definition 4 Given two proof systems $S_1 = (\Sigma_1, R_1, f_1)$ and $S_2 = (\Sigma_2, R_2, f_2)$, a proof procedure $\mathbf{P_1} = (S_1, g_1, F_1, P_1)$ and a polynomial simulation $s : C_{R_1} \mapsto C_{R_2}$, we call s-simulation of $\mathbf{P_1}$ the proof-procedure $\mathbf{P_2} = (S_2, g_2, F_2, P_2)$ such that (i) for all formulas A, $g_2(A) = s(g_1(A))$; (ii) for all proof-configurations $\pi \in C_{R_1}$, $F_2(s(\pi)) = s(F_1(\pi))$.

Now we can define the notion "S_2 is essentially more efficient than S_1" between proof systems.

Definition 5 We say that S_2 *outperforms* S_1 if there is a polynomial simulation s from S_1 to S_2 such that for every proof procedure $\mathbf{P_1}$ based on S_1, there is *no polynomial* p such that for all A

$$T(\mathbf{P_1}(A)) \leq p(T(\mathbf{P_2}(A))$$

where (i) P_2 is the s-simulation of P_1, (ii) $T(\mathbf{P_1}(A))$ and $T(\mathbf{P_2}(A))$ denote the respective time requirements of $\mathbf{P_1}$ and $\mathbf{P_2}$ on input A.

In other words, no proof procedure based on S_1 can p-simulate its s-simulation, so that the s-simulation itself is essentially more efficient than the simulated procedure. Notice that this is not the case for many of the separation results (expressed in terms of p-simulation) which are found in the literature. Then we are able to show that:

Proposition 1 *The analytic* **KE***-system outperforms the standard tableau system.*

3 Classical logic as the limit of a sequence of cut-bounded systems

We call $\mathbf{KE}(k)$ the system obtained from **KE** by allowing at most k analytic applications of the cut-rule. We show that:

- $\mathbf{KE}(0)$ has a decision procedure which runs in time $O(n^2)$ and the set for which it is complete includes the horn-clause fragment of propositional logic (however $\mathbf{KE}(0)$ is not restricted to clausal form logic);
- for every fixed k, $\mathbf{KE}(k)$ has a polynomial time decision procedure.

It is obvious that the set of tautologies for which $\mathbf{KE}(k)$ is complete tends to TAUT as k tends to infinity. The crucial point is that low-degree cut-bounded systems are powerful enough for a wide range of applications. We also investigate how the sets of tautologies for which each $\mathbf{KE}$(k) is complete can be given an independent characterization.

References

[Ajt88] Miklos Ajtai. The complexity of the pigeonhole principle. In *Proceedings of the 29th Annual Symposium on the Foundations of Computer Science*, 1988. Preliminary version.

[Bib82] W. Bibel. *Automated Theorem Proving.* Vieweg, Braunschweig, 1982.

[D'A90] Marcello D'Agostino. Investigations into the complexity of some propositional calculi. PRG Technical Monographs 88, Oxford University Computing Laboratory, 1990.

[D'A92] Marcello D'Agostino. Are tableaux an improvement on truth tables? Cut-free proofs and bivalence. *Journal of Logic, Language and Information*, 1992. To appear.

[Mon88] Marco Mondadori. Classical analytical deduction. Annali dell'Università di Ferrara; Sez. III; Discussion paper series 1, Università di Ferrara, 1988.

Labelled Refutation Systems: a Case-Study. Extended abstract

Marcello D'Agostino and Dov M. Gabbay
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ
e-mail: dg,mda@doc.ic.ac.uk

26 November 1992

1 Introduction

A logical system is now perceived by many researchers as a structured consequence relation satisfying identity and surgical cut (see below). These researchers came to accept this view in response to systems of non-monotonic and practical reasoning arising in application areas. The general notion of a *structured consequence relation* is put forward in [Gab]. Our aim in this paper is to examine to what extent it is possible to develop tableau systems for these new consequence relations. The need for tableau systems for such logics is made urgent especially since many existing systems have no algorithmic formulations. The problem is not trivial in view of the fact that the known tableau systems for classical, intuitionistic and modal logics appear to be semantically based, while many of the new consequence relations are presented either proof-theoretically or even axiomatically, with no intuitive semantics, and hence the very notion of semantic tableaux for such logics may be in doubt.

Our strategy is to develop a notion of *labelled tableaux* where the labels come from the Lindenbaum algebra which may, in many cases, represent the new consequence relations. The proper framework for dealing with such tableaux in their full generality is LDS [Gab91], but in this paper we concentrate on substructural logics in order to provide a clear illustration of the methodology.

2 From consequence relations to semantics

We start from arbitrary consequence relations $\vdash$ between *sequences* of formulas and formulas, satisfying:

Identity $$A \vdash A$$

Surgical cut $$\frac{\Gamma \vdash A \quad \Delta, A, \Lambda \vdash B}{\Delta, \Gamma, \Lambda \vdash B}$$

We observe that the turnstile satisfies the conditions for a pre-ordering (i.e. reflexivity and transitivity). The additional axioms for a specific consequence relation can be divided in two groups:

- axioms describing properties of $\vdash$, called *structural rules*;
- axioms describing properties of logical operators, called *operational rules.*

Under certain circumstances, algebraic semantics for arbitrary consequence relations can be easily obtained by applying the Lindenbaum-Tarski method. One defines the equivalence relation $A \cong B$

iff $A \vdash B$ and $B \vdash A$, and observes that $\mathcal{F}/\cong$, where $\mathcal{F}$ is the set of all well-formed formulas of the language, is partially ordered by the relation

$$||A|| \leq ||B|| \quad \text{iff} \quad A \vdash B.$$

One then considers the equivalence classes $||A||$, $||B||$ etc. as the set of truth-values of a valuation system, and, provided $\cong$ is a congruence, define algebraic operations on this set corresponding to the logical operators defined by the operational rules of $\vdash$. This construction is in most cases a straightforward one. For instance, in the case of the substructural logics over the language $\{\otimes, \rightarrow, \neg\}$ one obtains an algebraic structure $S = (M, \circ, 1, 0, \rhd, ^{\perp}, \leq)$ where: $(M, \circ, 1)$ is a monoid partially ordered by $\leq$, 0 is a constant different from 1, $\rhd$ is a binary operation defined as:

$$x \rhd y = \max\{z | z \circ x \leq y\},$$

and $^{\perp}$ is a unary operation satisfying:

$$x \circ y \leq 0 \Longleftrightarrow x \leq y^{\perp}.$$

This basic structure is then augmented with additional conditions on the partial ordering $\leq$ corresponding to the allowed structural rules. For instance:

Condition	$x \circ y = y \circ x$	$x \circ x \leq x$	$x \leq x \circ x$	$x \circ y \leq x$
Structural rule	$\dfrac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$	$\dfrac{\Gamma, A, \Delta \vdash C}{\Gamma, A, A, \Delta \vdash C}$	$\dfrac{\Gamma, A, A, \Delta \vdash C}{\Gamma, A, \Delta \vdash C}$	$\dfrac{\Gamma, A, \Delta \vdash C}{\Gamma, A, B, \Delta \vdash C}$

Let $\vdash_j$ be a consequence relation defined by structural rules and operational rules, let $\leq_j$ the partial ordering corresponding to $\vdash_j$ (i.e. a partial ordering with additional conditions corresponding to the structural rules). Consider arbitrary valuations h of formulas over the elements of the resulting algebraic structure S (defined in the usual way, i.e. as homomorphisms of the algebra of formulas into S). We have that $A \vdash_j B$ iff $h(A) \leq_j h(B)$ for every valuation h. Algebraic semantics of this kind for a family of subsystems of Heyting's intuitionistic logic are given by Dôsen in [Dôs88]. Such "semantics" are just reformulations of the sequent axioms defining a consequence relation. Relational (possibile-world) semantics for subsystems of intuitionistic logic, in the style of the Kripke semantics, are also given by Dôsen in [Dôs89].

Given a possible-world semantics, it is not trivial to develop a well-behaved tableau method for it. Morevoer, there may be several versions of a possible-world semantics for a given logic which, though being mathematically equivalent, are not equivalent for the formulation of a well-behaved algorithmic system of deduction based on them. The main aim of this paper consists in providing a simple and elegant tableau method for the family of substructural logics weaker than classical logic. It turns out that this goal, as well as orienting the choice of the most appropriate semantic formulation, yields new completeness results. For instance, Dosen's treatment is restricted to logics weaker than intuitionistic logic and therefore does not include the classical-like involutive negation typical of Andersonn and Belnap's relevance logic and of Girard's linear logic. On the other hand, our tableaux deal with all the substructural logics weaker than *classical logic* (whether with an involutive or non-involutive negation) in a uniform framework.

3 Tableaux for substructural logics

We first describe a possible-world semantics for the family of logical systems consisting of the fragments $\{\rightarrow, \neg, \otimes, \oplus\}$ of the substructural logics weaker than classical logic, and then provide a *uniform labelled tableau method* in which the operational rules are the same for all logical systems and the difference between one system and the other is expressed only by the condition for closing a branch. The possible-world semantics we adopt as basis of our tableau method is given in the following definition (in the full version of the paper we show how this semantics is derived from the

Lindenbaum algebra by using a general methodology). We assume a 0-order language containing the binary operations $\otimes$ and $\rightarrow$ as well as the unary operation $\neg$ and possibly the constants $\top$ and $\bot$ (these are theoretically dispensable, but practically useful). By a *quasi-ordered monoid with* 0 we mean a structure $(M, \circ, 1, 0, \leq)$ where $(M, \circ, 1)$ is a monoid with identity 1, 0 is a constant different from 1, and $\leq$ is a quasi-ordering satisfying:

$$x \leq y \text{ and } v \leq z \Longrightarrow x \circ v \leq y \circ z.$$

Definition 1

1. Let Q be a quasi-ordered monoid. A *PW-valuation over* Q is a two-argument function $\mathcal{F} \times Q \mapsto \{T, F\}$, where $\mathcal{F}$ is the set of formulas of the language, satisfying the following conditions:

   (a) $v(A, x) = T$ and $x \geq y$ implies $v(A, y) = T$.

   (b) $v(A \rightarrow B, x) = T$ iff $\forall y, v(A, y) = T$ implies $v(B, x \circ y) = T$.

   (c) $v(A \otimes B, x) = T$ iff $\exists y, z,\ x \leq y \circ z$ and $v(A, y) = T$ and $v(B, z) = T$.

   (d) $v(\top, x) = T$ iff $x \leq 1$

   (e) $v(\bot, x) = T$ iff $x \leq 0$

   The quasi ordering of the monoid behaves like an *accessibility relation* between worlds.

2. We say that a point z in the valuation space Q of a PW-valuation is *A-maximal* if

$$v(A, z) = T \text{ and } (\forall x \in Q)(v(A, x) = T \Longrightarrow x \leq z).$$

   We say that a PW-valuation over Q is *regular* if

$$(\forall A \in \mathcal{F})(\exists z \in Q)(z \text{ is } A\text{-maximal})$$

3. A *PW-model* is a triple (W, g, v) such that W is a quasi-ordered monoid (with 0), v is a regular PW-valuation over W and g is a unary operation on W such that

$$(\forall x) x \circ g(x) \not\leq 0 \text{ and } (\forall x) v(\neg A, x) = T \Longleftrightarrow v(A, g(x)) = F.$$

4. A PW-model is *regular* if the valuation v is regular.

5. A PW-model is *classical* if g is an involution, that is $g(g(x)) = x$.

6. A sequent $A_1, \ldots, A_n \vdash B$ is *verified* in a PW-model (Q, g, v) if for all $x_1, \ldots, x_n \in Q$, $v(B, x_1 \circ \cdots \circ x_n) = T$ or $v(A_i, x_i) = F$ for some $i = 1, \ldots, n$. Otherwise we say that the sequent is *falsified.*

The operator $\oplus$ is defined as $\neg A \rightarrow B$ and arises naturally as the dual of $\otimes$ in logical systems with an involutive, classical-like negation. PW-models are classified according to the general properties satisfied by their quasi-orderings which, in turn, reflect the structural properties of the consequence relation (see section 2). So if we consider a particular class **M** of PW-models corresponding to a particular substructural consequence relation $\vdash$, we show that A sequent S is *valid* if and only if it is verified in all *regular* PW-models in **M**. If the consequence relation contains an involutive negation like classical and linear negation, than we can restrict our attention to the subclass of *classical* models in **M**, i.e. we exclude potential countermodels.

This semantics allows us to derive a uniform tableau method for substructural logics. The declarative units of this tableau method are no longer signed formulas but *labelled signed formulas* where the labels bring the semantics into the syntax. Labelled signed formulas are expressions of the form $TA : x$ or $FA : x$ where x is a term belonging to a given algebra of the labels. In this

semantic-oriented context, the intended meaning is $v(A,x) = T$ and $v(A,x) = F$. The development of this tableau method from the semantics raises several algorithmic problems especially in connection with the formulation of a *systematic* proof procedure like the one which is available for classical tableaux. In the full version of the paper we show how these problems can be elegantly dealt with in the particular semantic framework we have chosen. Here we just formulate the rules for the implication-negation fragment and show an interesting example to illustrate the method at work.

Implication rules

$$\frac{\begin{array}{l} TA \to B : x \\ TA : y \end{array}}{TB : x \circ y} \qquad \frac{FA \to B : x}{\begin{array}{l} TA : c \\ FB : x \circ c \end{array}}$$

Where c is a *new atomic* label. Whenever the model is *regular*, we can identify the new atomic label c in the rule for $FA \to B : x$ with the maximal point x at which A is true. So, for *regular* PW-models, the following additional rule is allowed:

$$\frac{\begin{array}{l} TA : c \\ FA \to B : x \end{array}}{FB : x \circ c}$$

provided that c is an atomic label. The rule allows us to *reuse* the same atomic label c, instead of introducing a new one, and is crucial for the algorithmic formulation of some logics.

These rules are the *universal rules* of implication and are valid for all the logics in the family.

Closure rule

Different logics are distinguished by the following rule for *closing a branch* which depends on the algebra of the labels:

$$\frac{\begin{array}{l} TA : x \\ FA : y \end{array}}{\times} \quad \textit{provided } x \geq y$$

Where $\times$ indicates closure and $\geq$ is the accessibility relation characteristic of the class of PW-models under consideration. Moreover, the following branching rule, which expresses the fact that our valuations are *bivalent*, is also valid:

Bivalence rule

$$\frac{}{TA : x \mid FA : x}$$

for every label x.

The rules for negation are as follows:

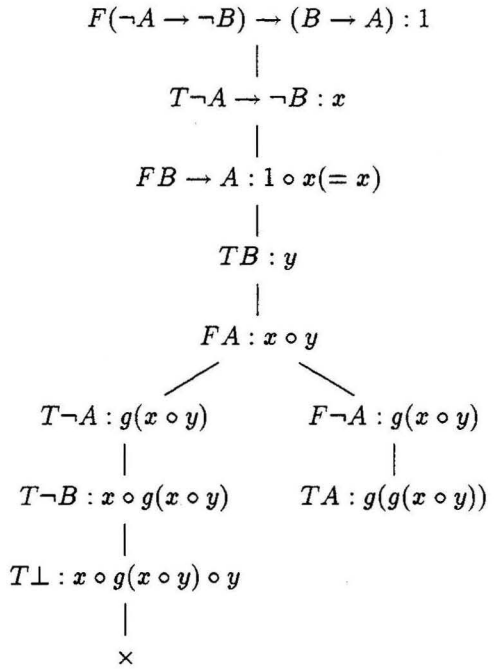
Negation rules

$$\frac{T\neg A : x}{FA : g(x)} \qquad \frac{F\neg A : x}{TA : g(x)} \qquad \frac{\begin{array}{l} T\neg A : x \\ TA : y \end{array}}{T\bot : x \circ y} \qquad \frac{}{F\bot : x \circ g(x)}$$

Observe that the first rule is redundant (it can be derived from the others). The last rule allows us to append to any branch the "axiom" $F\bot : x \circ g(x)$ which is always satisfied in any PW-model. In practice, this amounts to allowing the following closure rule:

$$\frac{T\bot : y}{\times} \quad \textit{provided } y \geq x \circ g(x) \textit{ for some } x.$$

Example 1 Consider the formula $(\neg A \to \neg B) \to (B \to A)$. This is a theorem of linear and relevance logic (as well as, of course, classical logic) but not a theorem of intuitionistic logic. We show that there are no counteremodels for the first two logics while showing at the same time that there is one for the third.



In this tree, the lefthand branch is closed for all the classes of PW-models satisfying $x \circ y = y \circ x$ (by the closure rule for $T\bot$ given above). The righthand branch is closed only for *classical* PW-models, in which the operation g is an involution.

In the paper we also introduce a technique for dealing with *labels containing variables.* Such variables are conveniently introduced when applying the branching rule given above, and their instantiation is postponed until a suitable value can be found which allows for closing a branch. Given a *potentially closed* tree, i.e. a tree such that all its branches contain at least a pair of labelled signed formulas of the form $TA : \sigma$, $FA : \nu$, where σ and ν are labelling terms containing variables, the problem of finding instantiations of all the variables which generate a closed tree for a given class of models amounts to solving systems of inequalities in a given algebra. In the paper we also describe a systematic procedure for dealing with variables in the labels and show how the use of variables hels avoiding backtracking in proof search.

References

[Dôs88] Kosta Dôsen. Sequent systems and grupoid models I. *Studia Logica*, 47:353–385, 1988.

[Dôs89] Kosta Dôsen. Sequent systems and grupoid models II. *Studia Logica*, 48:41–65, 1989.

[Gab] Dov M. Gabbay. General theory of structured consequence relations. In P. Schroeder Heister and Kosta Dôsen, editors, *Substructural Logics.* Oxford University Press. To appear.

[Gab91] D.M. Gabbay. Labelled deductive systems, part I. Technical Report CIS-Bericht-90-22, CIS-Universität München, February 1991. Preliminary partial draft of a book intended for Oxford University Press.

Proof Search Methods in Linear Logic

- abstract -

D. Galmiche

CRIN-CNRS & INRIA Lorraine
Campus Scientifique - B.P. 239
54506 Vandœuvre-les-Nancy Cedex
France
e-mail: galmiche@loria.fr

Abstract

In this paper, we present some efficient proof search methods in linear logic. After analyzing the reasons of redundancy in the linear sequent calculus, mainly identified as non-permutability of inferences and irrelevance, we propose a first method for proof search in such sequent calculus that is based on the non-permutability properties. From possible up and down inference movements in a proof, we can define sub-classes of proofs that are complete and tractable, and a notion of normal proof in linear logic that leads to an adequate method for constructing proofs for linear sequents.
After this investigation of the search space properties induced by the sequent calculus, we propose an other method for proof search based on matrix techniques, that is connected to proof nets construction.

1 Introduction

Linear logic is a powerful and expressive logic with connections to a variety of topics in computer science. We are mainly interested by the significance it may have in domains as logic programming or program synthesis through theorem proving and thus we focus on proof search in this framework. As a matter of fact, linear logic (denoted here LL) is a logic of actions introducing notions like controlled and strict resource management [8]. It disallows both weakening and contraction in general although they are introduced for local use through modalities and linear logic conserves a constructive character with a deep symmetry. Linear logic can be an appropriate framework to study logic programming (better than intuitionistic or classical logic) [11], concurrent aspects in logic programming [1], or plan generation [4, 13]. The main point is the resource-sensitive aspect of this logic. In previous works, we have considered the synthesis of correct programs using a theorem proving approach in constructive logics with extraction of programs from proofs [5]. Based on intuitionistic logics , they can present some limits due to the non-symmetrical character of such logics. Thus, it appears interesting to consider this approach in linear logic through an appropriate λ-calculus as logical language [12].
But in fact the first point is to understand what a proof or a proof net is in the various fragments of LL and also to be able to construct efficiently proofs of linear logic formulas. Hence, a theorem prover in linear logic can be a first step towards some effective

applications of LL based on the development of proofs like logic programming. For efficiency reasons, the construction of a linear logic prover imposes of course restrictions to an adequate fragment of LL [10]. Even if, it is not the central point, permutability properties is a basis of these different approaches. It is a classical concept in works on the conception of efficient proof search methods in non-classical logics [14, 16]. The cases of non-permutability amongst the rules that arise already in the propositional fragment of LL, serve to illustrate that the cut-free sequent calculus suffers from different classes of redundancy as in other calculi. A first kind of redundancy (notational redundancy) arises from the basic sequent calculus, a second one (irrelevance) arises from an emphasis of connectives and the third one (non-permutability), and the most interesting, arises from the semantics of the connectives [16]. At first, we propose to consider the inference permutability properties in linear logic aiming to efficient proof construction mechanization. A first attempt in this direction has been developed in [6] with an algorithm for automated deduction in the multiplicative and additive fragment of LL. Thus, after having systematically studied the permutability properties of inferences in LL, we can define the notion of up and down movements of inferences in a proof of LL and then we analyze the redundancy reduction in a proof. Thus, we are able to propose a new form of proof, called normal form and thus proof sub-classes that are complete and tractable.
After this investigation of the permutability properties, we can propose another approach based on matrix techniques (connections and matings [2, 3]) accordingly. At first, let us remark that there are connections between proof nets and proof-search algorithms in the matrix methods. In a previous work, we have proposed an algorithm for constructing automatically proof nets in linear logic [7] that appears to have strong relationship with connection methods. It constitutes a first step to have a matrix characterisation of linear logic. Even if the proof normalization solves the third type of redundancy, it could be interesting to treat also the other ones by a matrix method based on notions like path, connection, complementarity. After considering these different notions for linear logic, we analyze this method based on matrices for an efficient proof search in some fragments of linear logic, knowing that a main point here is to construct effectively a proof that can be used for applications mentionned above.

2 Linear logic

Linear logic (LL) has been introduced by Girard [8] as a logic of actions being more expressive than traditional logics (classical or intuitionistic ones). Characterized by the absence of structural rules (contraction and weakening) and by a specific treatment of the negation, LL has proofs that can be considered as actions and introduces a dynamical resource management in these proofs. In LL, conjunction and disjunction are split into multiplicative $(\otimes, \wp)$ and additive $(\&, \oplus)$ versions and the modal operators ? and ! allow to restore the power of classical logic. We refer the reader to [8] for a broad explanation of the purpose and the meaning of linear logic.

3 Permutability and normalization

We start with the study of the *inference permutability* notion in linear logic. It is a basic and important concept for developing efficient proof search methods [14]. After having defined the permutation notion for two inferences, we can give, through theorems, the different cases of permutability. For example, we have $\oplus$ and $\otimes$ that are permutable but even if $\&$ can permute with $\oplus$ the reverse is not true, i.e., $\oplus$ does not permute with $\&$.

From this analysis, it is interesting to define the notion of (up and down) *movements* in a proof of LL. Thus, we prove that we can move up in a proof the inferences of type $\otimes, \oplus, \exists$ and move down $\wp, \&, \forall$ and we define proof transformations by moving inferences up or down in a given proof. From these movement possibilities and this classification between the connectives that is similar to another one [1] based on synchronization characteristics, we can define proof sub-classes that will be complete and tractable (irrelevant choices in proof search are cancelled) and the concept of *normal proof* with the result that, for any proof Π of LL there exists a normal proof Π_n with the same conclusion. In a goal-oriented approach we can define a reduction order of the goal and theorem proving will consist in constructing a normal proof. For efficiency reasons, it appears necessary to restrict the study to particular fragments of LL adapted to applications as logic programming. Moreover, to treat the various redundancy properties, it could be interesting to consider other methods, not directly based on sequent calculus, like matrices techniques or resolution.

4 Proof nets and matrix method

When we try to develop algorithms for proof nets construction in multiplicative fragment of LL we observe relationships with proof-search algorithms based on matrixes and connections. The algorithm of [7] was a first step that helps us to understand these relations and to consider a matrix characterisation of validity in linear logic. The plan generation is an application topic that has already illustrated the strong relationship between refinements of connection method and construction of proof net in LL [4, 13]. Moreover, this interest of such approach is to suppress the various sources of redundancy in proof search in LL.
To develop this point, we need a notion of *polarity* which allows to identify antecedent subformulae in terms of the formula structure whose validity we are testing. It is different of the other one introduced by Girard in [9] that could also help to have a good treatment of the proof nets. We expect to formulate such a characterisation in terms of *position* in a *formula tree*, after giving the definition of signed formulas (pair $< A, n >$ with A formula and $n \in \{0, 1\}$) and specifying their type: for example, $< A \otimes B, 1 >$ and $< A \wp B, 0 >$ are of conjunctive type, $< A \otimes B, 0 >$ and $< A \wp B, 1 >$ are of disjunctive type. With the notion of *path* through a formula defined as specific subset of the position set and the one of *connection* as a subpath consisting of two opposed atomic positions, we can prove formulas in some fragments of LL with a matrix method that completes the first part on our study based on sequent calculus and on normalisation. It confirms the specific treatment of the different connectives of LL during theorem proving and the relationship with proof net construction in this framework.

Other methods based on resolution have been proposed emphasizing also the importance of the non-permutability properties [10, 15]. The interest of some comparisons is emphasized by the fact that a matrix characterisation of a logic enables the application of resolution search strategies to proof search [16].
Here, after a systematic study of the logical properties of the linear logic framework, undependentely of the possible applications, our aim is to have efficient and automated methods for proofs and proof nets construction for a further application of the paradigm *proofs as programs* in linear logic based frameworks.

References

[1] J.M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3), 1992.

[2] P.B. Andrews. On connections and higher-order logic. *Journal of Automated Reasoning*, 5:257–291, 1989.

[3] W. Bibel. On matrices with connections. *Journal of ACM*, 28(4):633–645, 1981.

[4] B. Fronhöfer. Linear proofs and linear logic. In *European Workshop JELIA '92, LNAI 633*, pages 106–125, 1992.

[5] D. Galmiche. Constructive system for automatic program synthesis. *Theoretical Computer Science*, 71(2):227–239, 1990.

[6] D. Galmiche and G. Perrier. Automated deduction in additive and multiplicative linear logic. In *Logic at Tver '92, Logical Foundations of Computer Science Symposium, LNCS 620*, pages 151–162, Tver, Russia, July 1992.

[7] D. Galmiche and G. Perrier. A procedure for automatic proof nets construction. In *LPAR'92, International Conference on Logic Programming and Automated Reasoning, LNAI 624*, pages 42–53, St. Petersburg, Russia, July 1992.

[8] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[9] J.Y. Girard. On the unity of logic. Research report 1467, Inria, June 1991.

[10] J. Harland and D. Pym. On resolution in fragments of classical linear logic. In *LPAR'92, International Conference on Logic Programming and Automated Reasoning, LNAI 624*, pages 30–41, St. Petersburg, Russia, July 1992.

[11] J.S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In *6th IEEE Symposium on Logic in Computer Science*, pages 32–42, Amsterdam, The Netherlands, July 1991.

[12] P. Lincoln and J. Mitchell. Operational aspects of linear lambda calculus. In *7th IEEE Symposium on Logic in Computer Science*, pages 235–246, Santa-Cruz, California, June 1992.

[13] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic. In *Foundations of Software Technology and Theoretical Computer Science, LNCS 472*, pages 63–75, Bangalore, India, December 1990.

[14] N. Shankar. Proof search in the intuitionistic sequent calculus. In *11th Conference on Automated DEduction, LNAI 607*, pages 522–536, Saratoga Springs, June 1992.

[15] T. Tammet. Proof search in linear logic. Submitted to Journal of Automated Reasoning, 1992.

[16] L.A. Wallen. *Automated Proof search in Non-Classical Logics.* MIT Press, 1990.

NON–LOOPING TABLEAU FOR S4

Roderic A. Girle

School of Computing and Information Technology
Griffith University
Nathan Queensland 4111
AUSTRALIA

INTERNET: rag@cit.gu.edu.au

ABSTRACT

Standard semantics for **S4** do not automatically lead to the production of finite counter–models for non–theorems. When the standard semantics is coded into a Fitting style tableau theorem prover the prover goes into a non–terminating loop when given certain non–theorems. In this paper we set out a semantic tableau system for **S4** in which the tableau terminate for both the theorems and the non–theorems. The classical looping problem is dealt with by decomposition rules which take advantage of the finite number of modalities in **S4**.

1. PRELIMINARIES

S4 has the finite model property (McKinsey [1941]). This means that if a formula is not a theorem, then it is possible to produce a finite counter–model. It also means that there should be a refutation theorem prover, such as a tableau prover, which will be able to produce a finite counter–model to any non–theorem. Nevertheless, most standard tableau provers do not easily produce counter–models. Fitting style provers for **S4**, which embody the standard Hintikka semantics for **S4**, go loopy when confronted with formulas such as:

$$\Box\Diamond p \rightarrow \Diamond\Box p$$

While working recently on a Fitting style prover for **S4**, some unexpected consequences flowed from two changes to an otherwise plain tableau prover. The first change was made when it became clear that there would be some considerable increase in efficiency if formulas were re-written so reduce sequences of modal operators of the same kind to a single operator. For example, before a formula such as:

1. $\Box p \rightarrow \Box\Box\Box p$

is tested, it can be rewritten to:

2. $\Box p \rightarrow \Box p$

It is straightforward to prove *2*.

The second change is not easy to explain unless one is already familiar with both tableau provers for modal logic and with Fitting style provers. Nevertheless, the change can be descibed by means of a contrast. Before the change, whenever the prover decomposed a formula of the form $\Box\,\alpha$ by means of the usual reflexive decomposition rule, it simply left the formula available for repeated decomposition. But, when

the prover was changed, it was forced to mark any formula of the form $\Box \alpha$ as not available for decomposition once it had been dealt with by the usual reflexive decomposition rule, and the use of this rule was delayed as long as possible in the prover's strategy.

The unusual consequences which flowed from these two changes concern the way in which tableau provers usually go into an infinite loop. The non-theorem, *3*, is a typical formula which sends a plain tableau prover into a loop.

3. $\Box \Diamond p \rightarrow \Diamond \Box p$

The prover with the two changes did not go into a loop when presented with *3*. The prover quickly and non-loopingly responded that the formula was not a theorem.

This immediately raised the spectre of non-soundness. I consulted various logicians. Kit Fine discovered a counter-example to soundness. The counter-example was *4.* which is a theorem and was rejected by the prover:

4. $\Box (p \ \& \ \Diamond \sim p) \rightarrow \sim \Diamond q$

It turned out that the prover was not able to distribute the $\Box$ over conjunction in certain crucial contexts. The prover had no problems with the simple distribution theorem:

5. $\Box (p \ \& \ q) \rightarrow (\Box p \ \& \ \Box q)$

So, the discovery of the counter–example involved working through some tableau and discovering the contexts in which it would fail to make the appropriate distribution as it decomposed the formulas.

Once this problem was detected, a third change was introduced into the prover. Necessitated conjunctions were re–written by replacing them with their necessitated conjuncts. Of course, the question of soundness was still lurking in the wings.

We now show that **S4** is sound and complete with respect to the decompositional semantics expressed in the prover.

2.0 FORMALITIES

Given the usual vocabulary for **S4** with the unary connectives: $\sim$ and $\Box$, and the binary connectives $\rightarrow$ and $\&$, and the usual definitions of $\vee$ and $\Diamond$, and the formation rules for **S4**, we set out the axiomatics and the semantics.

2.1 Axiomatics

The axiomatics, in Lemmon style, are as follows:

1. $(A \rightarrow (B \rightarrow A))$
2. $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)))$
3. $((\sim A \rightarrow \sim B) \rightarrow (B \rightarrow A))$
4. $(\Box A \rightarrow A)$
5. $(\Box (A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B))$
6. $(\Box A \rightarrow \Box \Box A)$

R1. $\vdash A, \vdash (A \rightarrow B) \Rightarrow \vdash B$
R2. $\vdash A \Rightarrow \vdash \Box A$

We assume the usual definitions of *proof*, *theorem*, and *deduction from a set of premises*, and the *Deduction Theorem*.

2.2 Semantics

We now set out Hintikka style model–system semantics for **S4**. Although these are essentially decompositional, they need modification if looping tableau are to be avoided. We will set out modifications later. For the moment we set out the unmodified semantics so that they represent fairly closely the Fitting style **S4** prover, and so that they can be applied in Modal Truth–trees.

A model–system is a set, Ω, of model–sets, μ_i:

$$\Omega = \{\mu_0, \ldots, \mu_i, \ldots\}$$

It is useful to define:

$$\mathbf{Nec}'(\mu) = \{\Box A \mid \Box A \in \mu\}$$

$$\mathbf{Denec}'(\mu) = \{A \mid \Box A \in \mu\}$$

We note that if μ is finite then both $\mathbf{Nec}'(\mu)$ and $\mathbf{Denec}'(\mu)$ will be finite.

We also use [ ] for any finite sequence of zero or more unary connectives.

Model–sets are sets of formulas which conform to the following *consistency conditions*:

(C.∅) $\{A, \sim A\}$ *is not a sub–set of any model–set.*
(C.~~) *If* $\{[\ ]\sim\sim A\} \subseteq \mu_i$ *then* $\{[\ ]A\} \subseteq \mu_i$.
(C.&) *If* $\{(A \,\&\, B)\} \subseteq \mu_i$ *then* $\{A, B\} \subseteq \mu_i$.
(C.~&) *If* $\{\sim(A \,\&\, B)\} \subseteq \mu_i$ *then either* $\{\sim A\} \subseteq \mu_i$ *or* $\{\sim B\} \subseteq \mu_i$.
(C.→) *If* $\{(A \rightarrow B)\} \subseteq \mu_i$ *then either* $\{\sim A\} \subseteq \mu_i$ *or* $\{B\} \subseteq \mu_i$.
(C.~→) *If* $\{\sim(A \rightarrow B)\} \subseteq \mu_i$ *then* $\{A, \sim B\} \subseteq \mu_i$.
(C.~□) *If* $\{[\ ]\sim\Box A\} \subseteq \mu_i$ *then* $\{[\ ]\Diamond \sim A\} \subseteq \mu_i$.
(C.~◇) *If* $\{[\ ]\sim\Diamond A\} \subseteq \mu_i$ *then* $\{[\ ]\Box \sim A\} \subseteq \mu_i$.
(C.◇) *If* $\{\Diamond A\} \subseteq \mu_i$ *then there is some set,* μ_k, *such that* $\mu_k = \mathbf{Nec}'(\mu_i) \cup \{A\}$
(C.□) *If* $\{\Box A\} \subseteq \mu_i$, *then* $\{A\} \subseteq \mu_i$.

A formula, *A*, is said to be *satisfiable* iff there is some model–system in which there is a model–set, μ, such that $\{A\} \subseteq \mu$.

A formula, *A*, is *Valid* iff $\sim A$ is not satisfiable.

These definitions can be applied to the testing of formulas in modal truth–trees. We adopt the convention that:

$$A \quad (i) \qquad \text{means } \{A\} \subseteq \mu_i.$$

For example, consider the axiom schema:

$$(\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B))$$

We set out the truth–tree as follows:

1.	$\sim(\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B))$	(i)
2.	$\Box(A \rightarrow B)$	(i) from 1, (C.$\sim\rightarrow$)
3.	$\sim(\Box A \rightarrow \Box B)$	(i) from 1, (C.$\sim\rightarrow$)
4.	$\Box A$	(i) from 3, (C.$\sim\rightarrow$)
5.	$\sim \Box B$	(i) from 3, (C.$\sim\rightarrow$)
6.	$\Diamond \sim B$	(i) from 5, (C.$\sim \Box$)
7.	$\sim B$	(j) from 6, (C.$\Diamond$)
8.	$\Box A$	(j) from 6, 4, (C.$\Diamond$)
9.	$\Box(A \rightarrow B)$	(j) from 6, 2, (C.$\Diamond$)
10.	A	(j) from 8, (C.$\Box$)
11.	$(A \rightarrow B)$	(j) from 9, (C.$\Box$)
12.	$\sim A$ (j) $\times$ $\quad$ B (j) $\times$	from 11, (C.$\rightarrow$)

So it is a theorem.
Even more importantly, consider the formula:

$$(\Box \Diamond A \rightarrow \Diamond \Box A)$$

The typical looping truth–tree is:

1.	$\sim(\Box \Diamond A \rightarrow \Diamond \Box A)$	(i)
2.	$\Box \Diamond A$	(i) from 1, (C.$\sim\rightarrow$)
3.	$\sim \Diamond \Box A$	(i) from 1, (C.$\sim\rightarrow$)
4.	$\Box \sim \Box A$	(i) from 3, (C.$\sim\Diamond$)
5.	$\Box \Diamond \sim A$	(i) from 4, (C.$\sim \Box$)
6.	$\Diamond \sim A$	(i) from 5, (C.$\Box$)
7.	$\sim A$	(j) from 6, (C.$\Diamond$)
8.	$\Box \Diamond A$	(j) from 6, 2, (C.$\Diamond$)
9.	$\Box \Diamond \sim A$	(j) from 6, 5, (C.$\Diamond$)
10.	$\Diamond A$	(j) from 8, (C.$\Box$)
11.	A	(k) from 9, (C.$\Diamond$)
12.	$\Box \Diamond A$	(k) from 10, 8, (C.$\Diamond$)
13.	$\Box \Diamond \sim A$	(k) from 10, 9, (C.$\Diamond$)
	⋮	

So we have the usual continuation to an infinite truth–tree.

2.3 Decompositional Semantics

In order to avoid the sort of thing which happened in the last truth–tree we modify our definition of a model–system of model–sets of formulas. Our modification sets up a *Model–sequence*, Ω, of sets, Π_i, of model–systems. A Model–sequence is a *Tree*. Each Π_i is a *path–set* in a tree. The model–sequence begins with a set Π_0, $\Pi_0 = \{\Omega_0\}$, where Ω_0 is a model–system of a single model–set containing the negation of the formula to be tested for theoremhood (or a finite set of formulas being tested for inconsistency). The sets of model–systems in the sequence contain the systems which are 'generated' by decomposing formulas or sets of formulas in one of the

model–sets in one of the model–systems in the previous set of model–systems. Formulas are *always* removed as they are decomposed. This means that even necessitated formulas can be decomposed only once. We define a decomposition relation which orders the sets of model–systems in a model–sequence. In this way we limit the model–sets to finite sets of formulas.

There are several substantial modifications in the definitions set out below to the standard modal truth–trees. There are four which should be noted:

(a) The contraction of iterated unary modal operators.
(b) The distribution of necessity over conjunction.
(c) The once only decomposition of formulas, including those with a necessity main operator.
(d) Allowing for inconsistent model–sets.

Definitions:

A model–set is *inconsistent* iff it contains a formula and its negation.
A model–set is *consistent* iff it is not inconsistent.
A *literal model–set* is a model–set of literals.
A model–system is *inconsistent* iff all its model–set members are inconsistent.
A model–system is *consistent* iff it is not inconsistent.
A *literal model–system* is a model–system of literal model–sets.

We define a Model–sequence, Ω, to be a sequence of finite sets of finite model–systems:

$$\Omega = <\Pi_0, \Pi_1, \ldots, \Pi_n, \ldots>$$

ordered by the Decomposition Relation. We now define the Decomposition Relation:

$$\textbf{Decomp}'(\{B_1, \ldots, B_k\}, \mu_i, \Omega_n, \Pi_x) = \Pi_y$$

The relation maps from a set of model–systems, Π_x, to a set of model–systems, Π_y, by virtue of a set of formulas $\{B_1, \ldots, B_k\}$, which is a sub–set of μ_i which is a member of Ω_n in Π_x. The difference between Π_x and Π_y is that Ω_n is replaced by Ω_m, and in some cases, a set Ω_z is added. Π_x is known as the *originating set of model–systems* and Π_y is known as the *resultant set of model–systems*. The set of formulas $\{B_1, \ldots, B_k\}$ is known as the *generating* set of formulas.

(Decomp Lit) *If* μ_i *is a* literal model–set, *then* $\textbf{Decomp}'(\mu_j, \mu_i, \Omega_n, \Pi_x) = \Pi_x$

(Decomp ~ ~) $$\textbf{Decomp}'(\{[\ ]\sim\sim B\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{[\ ]\sim\sim B\}) \cup \{[\ ]B\}\})\}$$

(Decomp &) $$\textbf{Decomp}'(\{(B \,\&\, C)\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{(B \,\&\, C)\}) \cup \{B, C\}\})\}$$

(Decomp ~&) $$\textbf{Decomp}'(\{\sim(B \,\&\, C)\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{\mu_i - \{\sim(B \,\&\, C)\}) \cup \{\sim B\}\})\} \cup \{((\Omega_n - \{\mu_i\}) \cup \{\mu_i - \{\sim(B \,\&\, C)\}) \cup \{\sim C\}\})\}$$

(Decomp→) $\mathbf{Decomp}'(\{(B \rightarrow C)\}, \mu_i, \Omega_n), \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{\mu_i - \{(B \rightarrow C)\}) \cup \{\sim B\}\})\} \cup \{((\Omega_n - \{\mu_i\}) \cup \{\mu_i - \{(B \rightarrow C)\}) \cup \{C\}\})\}$

(Decomp ~→) $\mathbf{Decomp}'(\{(B \,\&\, C)\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{(B \rightarrow C)\}) \cup \{B, \sim C\}\})\}$

The definitions above are the effective account of standard propositional logic truth–trees. The definitions below are for the modal logic.

(Decomp ~◊) $\mathbf{Decomp}'(\{[\ ]\sim\Diamond B\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{[\ ]\sim\Diamond B\}) \cup \{[\ ]\Box \sim B\}\})\}$

(Decomp ~□) $\mathbf{Decomp}'(\{[\ ]\sim \Box B\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \mu_i) \cup \{(\mu_i - \{[\ ]\sim \Box B\}) \cup \{[\ ]\Diamond \sim B\}\})\}$

The next three definitions cover the first two substantial modifications of the usual account of modal truth–trees.

(Decomp □ □) $\mathbf{Decomp}'(\{\Box\ \Box B\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{\Box\ \Box B\}) \cup \{\Box B\}\})\}$

(Decomp ◊◊) $\mathbf{Decomp}'(\{\Diamond\Diamond B\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{\Diamond\Diamond B\}) \cup \{\Diamond B\}\})\}$

(Decomp □ &) $\mathbf{Decomp}'(\{\Box (B \,\&\, C)\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{\Box (B \,\&\, C)\}) \cup \{\Box B, \Box C\}\})\}$

The last two definitions are similar to the usual modal truth–tree definitions.

(Decomp ◊) $\mathbf{Decomp}'(\{\Diamond B_1, \ldots, \Diamond B_k\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{\{B_1\} \cup \mathbf{Nec}'(\mu_i), \ldots, \{B_k\} \cup \mathbf{Nec}'(\mu_i)\})\}$

(Decomp □) $\mathbf{Decomp}'(\{\Box B_1\}, \mu_i, \Omega_n, \Pi_x) = (\Pi_x - \{\Omega_n\}) \cup \{((\Omega_n - \{\mu_i\}) \cup \{(\mu_i - \{\Box B\}) \cup \{B\}\})\}$

The last of the definitions covers the third of the substantial modifications of the usual account of modal truth–trees. In the standard truth–trees it is possible to repeatedly decompose necessitated formulas. This is not so under the above definitions.

Two additional things are worth noting. First, that a literal model–system has to be the last one in any Model–sequence. Second, that there is only one clause in the definition of **Decomp** which might result in the generation of more sets for any resultant Ω_j than were in the originating Ω_i. It is **(Decomp ◊)**. Also, any of the sets in the resultant model–system which are different to the sets in the originating model–system can be referred to as the *generated* sets.

Definitions:

A finite Model–sequence is *consistent* iff it has at least one consistent literal model–system as one of its final members.

A finite set of formulas, μ, is *satisfiable* iff $\Pi_0 = \{\Omega_0\} = \{\{\mu\}\}$ and Ω_0 is the first model–system in at least one consistent finite Model–sequence.

A formula, *A*, is *Valid* iff $\{\sim A\}$ is not satisfiable.

An argument, $\Gamma \vdash A$, is *Valid* iff $\Gamma \cup \{\sim A\}$ is not satisfiable.

We now show how these definitions apply to some formulas. As might be expected, we get something just like modal truth–trees.

For example, consider the formula:

$$(\Box \Diamond A \rightarrow \Diamond \Box A)$$

We set out the Model–sequence as follows:

1.	$\{\{\{\sim(\Box \Diamond A \rightarrow \Diamond \Box A)\}\}\}$	$= \Pi_0$	
2.	$\{\{\{\Box \Diamond A, \sim\Diamond \Box A\}\}\}$	$= \Pi_1$	*1*, **(Decomp** $\sim\rightarrow$**)**
3.	$\{\{\{\Box \Diamond A, \Box \sim \Box A\}\}\}$	$= \Pi_2$	*2*, **(Decomp** $\sim\Diamond$**)**
4.	$\{\{\{\Box \Diamond A, \Box \Diamond \sim A\}\}\}$	$= \Pi_3$	*3*, **(Decomp** $\sim\Box$**)**
5.	$\{\{\{\Box \Diamond A, \Diamond \sim A\}\}, \{\{\Diamond A, \Box \Diamond \sim A\}\}\}$	$= \Pi_4$	*4*, **(Decomp** $\Box$**)**
6.	$\{\{\{\Box \Diamond A, \sim A\}\}, \{\{A, \Box \Diamond \sim A\}\}\}$	$= \Pi_5$	*5*, **(Decomp** $\Box$**)**
7.	$\{\{\{\Diamond A, \sim A\}\}, \{\{\Diamond \sim A, A\}\}\}$	$= \Pi_6$	*6*, **(Decomp** $\Diamond$**)**
8.	$\{\{\{A\}\}, \{\{\sim A\}\}\}$	$= \Pi_7$	*7*, **(Decomp** $\Diamond$**)**

Since we have arrived at all consistent literal model–systems, the formula is not Valid. We now consider Fine's formula and its Model–sequence:

1.	$\{\{\{\sim(\Box(A \,\&\, \Diamond \sim A) \rightarrow \sim\Diamond B)\}\}\}$	$= \Pi_0$	
2.	$\{\{\{\Box(A \,\&\, \Diamond \sim A), \sim\sim\Diamond B\}\}\}$	$= \Pi_1$	*1*, **(Decomp** $\sim\rightarrow$**)**
3.	$\{\{\{\Box(A \,\&\, \Diamond \sim A), \Diamond B\}\}\}$	$= \Pi_2$	*2*, **(Decomp** $\sim\sim$**)**
4.	$\{\{\{\Box A, \Box \Diamond \sim A, \Diamond B\}\}\}$	$= \Pi_3$	*3*, **(Decomp** $\Box\,\&$**)**
5.	$\{\{\{B, \Box A, \Box \Diamond \sim A\}\}\}$	$= \Pi_4$	*4*, **(Decomp** $\Diamond$**)**
6.	$\{\{\{B, \Box A, \Diamond \sim A\}\}\}$	$= \Pi_5$	*5*, **(Decomp** $\Box$**)**
7.	$\{\{\{\sim A, \Box A\}\}\}$	$= \Pi_6$	*6*, **(Decomp** $\Diamond$**)**
8.	$\{\{\{\sim A, A\}\}\}$	$= \Pi_7$	*6*, **(Decomp** $\Box$**)**

So it is Valid also. It should be noted that if **(Decomp** $\Box\,\&$**)** had not been used then we would not have arrived at an inconsistent literal model–system.

3. Completeness

We now begin the completeness proof for **S4**. There are three main sections to the proof. First, we define proof theoretic consistency for **S4**, and prove some Lemmas about sets of formulas. Second, we set out the decompositional semantics of the tableau prover by modifying the model–system semantics set out above. Third, we show that if *A* is a *Valid* formula in the semantics then it is a theorem of **S4**.

3.1 Proof Theoretic Consistency

We first define, in typical fashion, *proof–theoretic consistency*, or **p**–*consistency*, following Hunter (pg 107):

> *A finite set,* $\{A_1, \ldots, A_n\}$, *of formulas of* **S4** *is* **p**–*consistent iff there is no formula B such that* $\{A_1, \ldots, A_n\} \vdash_{S4} (B \ \& \ \sim B)$

conversely:

> *A finite set,* $\{A_1, \ldots, A_n\}$, *of formulas of* **S4** *is* **p**–*inconsistent iff there is some formula B such that* $\{A_1, \ldots, A_n\} \vdash_{S4} (B \ \& \ \sim B)$

It follows directly from these definitions and the definition of a proof that any set μ, where $\{\sim A, A\} \subseteq \mu$, is a **p**–inconsistent set. The converse result can be the first Lemma:

Lemma 1: *If* μ *is* **p**–*consistent, then there is no formula, A, such that* $\{A, \sim A\} \subseteq \mu$.

It also follows from the definition of a **p**–inconsistent set and the *ex falso quodlibet* theorem that:

Lemma 2: *If a finite set,* μ, *is* **p**–*inconsistent then, for every formula A,* $\mu \vdash_{S4} (A \ \& \ \sim A)$

Proofs of the following are standard.

Lemma 3: *If* μ *is finite and* $\mu = \{A_1, \ldots, A_n\}$ *and is* **p**–*inconsistent, then* $\vdash_{S4} \sim(A_1 \ \& \ \ldots \ \& \ A_n)$

Lemma 4: *If* μ *is finite and contains only literals and is* **p**–*inconsistent, then for some literal,* $\sim A$: $\{\sim A, A\} \subseteq \mu$

Lemma 5: *If* μ *is* **p**–*consistent and* $\mu \vdash_{S4} A$ *then* $\mu \cup \{A\}$ *is* **p**–*consistent.*

Lemma 6: *If* μ *is* **p**–*consistent and* $\{\sim(A \ \& \ B)\} \subseteq \mu$ *then either* $\mu \cup \{\sim A\}$ *or* $\mu \cup \{\sim B\}$ *is* **p**–*consistent.*

Lemma 7: *If* $\{A, \Box B_1, \ldots, \Box B_n\}$ *is* **p**–*inconsistent, then so is* $\{\Diamond A, \Box B_1, \ldots, \Box B_n\}$

Lemma 8: *If* $\{A, \Box B_1, \ldots, \Box B_n\}$ *is* **p**–*inconsistent, then so is* $\{\Box \Diamond A, \Box B_1, \ldots, \Box B_n\}$

We now prove the crucial Lemma:

Lemma 9: *If* $\Pi_0 = \{\{\mu\}\}$ *and* μ *is a* **p**–consistent *set of formulas, then there is a* consistent Model–sequence *of which* Π_0 *is the first member.*

The proof is by induction on the length of a Model–sequence beginning with Π_0:

$$<\Pi_0, \ldots, \Pi_{n-1}, \Pi_n, \ldots, \Pi_i> \qquad (\text{where } 0 \leq n \leq i)$$

Basis: There is only one element in the Model–sequence. So $\Pi_i = \Pi_0$. So, $\Pi_i = \{\Omega_0\} = \{\{\mu\}\}$. If μ is **p**–consistent, then it does not contain any formula and its negation. (Lemma 1) Since there is only one model–system in the

Model–sequence, μ contains only literals, and Ω_0 is a consistent literal model–system, $<\Pi_i>$ is a consistent Model–sequence.

Induction: There are two parts to the induction.
FIRST, there are eleven sub–cases, one for each clause of the definition of **Decomp**, in which it is established that the **Decomp** relation will map from **p**–consistent to **p**–consistent sets.

SECOND, it needs to be established that the Model–sequence terminates. This will be proved by considering the decrease in the number of connectives in the model–sets in the model–systems in the Model–sequence.

FIRST:

(1) **(Decomp** Lit) In this case $\Pi_{n-1} = \Pi_n$, so by inductive hypothesis Π_n is consistent.

(2) **(Decomp** ~ ~) This is a case of the replacement of formulas by their **S4** equivalents. Proof is straightforward.

(3) **(Decomp** &) In this and subsequent cases: $\Pi_{n-1} \neq \Pi_n$ and since by inductive hypothesis Π_{n-1} is consistent,
at least one model–system in Π_{n-1} is consistent. Let it be Ω_x

So, for some $\mu_i \in \Omega_x \in \Pi_{n-1}$ and $\{(B \,\&\, C)\} \subseteq \mu_i$:
Decomp'$(\{(B \,\&\, C)\}, \mu_i, \Omega_x, \Pi_{n-1}) = \Pi_n$, and
$\Pi_n = (\Pi - \{\Omega_x\}) \cup \{\Omega_y\}$ where
$\Omega_y =$
$\{((\Omega_x - \{\mu_i\}) \cup ((\mu_i - \{(B \,\&\, C)\}) \cup \{B, C\}))\}$
The only difference between Π_{n-1} and Π_n is that
μ_i has been replaced by $(\mu_i - \{(B \,\&\, C)\}) \cup \{B, C\}$
Since μ_i is consistent and contains $(B \,\&\, C)$ it follows from Lemma 5 that $\mu_i \cup \{B, C\}$ is consistent.
Since $\mu_i \cup \{B, C\}$ is consistent,
$(\mu_i \cup \{B, C\}) - \{(B \,\&\, C)\}$ is also.
So, Ω_y is consistent.
So, Π_n is consistent.

(4) **(Decomp** ~ &)
For some $\mu_i \in \Omega_x \in \Pi_{n-1}$ and $\{\sim(B \,\&\, C)\} \subseteq \mu_i$:
Decomp'$(\{\sim(B \,\&\, C)\}, \mu_i, \Omega_x, \Pi_{n-1}) = \Pi_n$, and
$\Pi_n = (\Pi_n - \{\Omega_x\}) \cup \{\Omega_y, \Omega_z\}$ where
$\Omega_y =$
$\{((\Omega_x - \{\mu_i\}) \cup \{\mu_i - \{\sim(B \,\&\, C)\}) \cup \{\sim B\}\})\}$ and
$\Omega_z =$
$\{((\Omega_x - \{\mu_i\}) \cup \{\mu_i - \{\sim(B \,\&\, C)\}) \cup \{\sim C\}\})\}$
Since μ_i is consistent and contains $\sim(B \,\&\, C)$ it follows from Lemma 6 that either $\mu_i \cup \{\sim B\}$ is consistent
or $\mu_i \cup \{\sim C\}$ is consistent.

So it follows that either $(\mu_i \cup \{\sim B\}) - \{\sim(B \,\&\, C)\}$
or $(\mu_i \cup \{\sim C\}) - \{\sim(B \,\&\, C)\}$ is consistent.
So, either Ω_y or Ω_z is consistent.
So, Π_n is consistent.

(5) (**Decomp** $\sim\Diamond$), (6) (**Decomp** $\sim\Box$), (7) (**Decomp** $\Box\Box$), (8) (**Decomp** $\Diamond\Diamond$), and (9) (**Decomp** $\Box$ &) are all cases of the replacement of formulas by their S4 equivalents. Proof is straightforward.

(10) (**Decomp** $\Diamond$)
For some $\mu_i \in \Omega_x \in \Pi_{n-1}$ and $\{\Diamond B_1, \ldots, \Diamond B_k\} \subseteq \mu_i$:
Decomp'$(\{\Diamond B_1, \ldots, \Diamond B_k\}, \mu_i, \Omega_x, \Pi_{n-1}) = \Pi_n$, and
$\Pi_n = (\Pi_n - \{\Omega_x\}) \cup \{\Omega_y$ where
$\Omega_y =$
$(\Omega_x - \{\mu_i\}) \cup \{\{B_1\} \cup \text{Nec}'(\mu_i), \ldots, \{B_k\} \cup \text{Nec}'(\mu_i)\}$
Let one of the sets which replaces μ_i in Ω_x to give Ω_y
be the set which contains B_1, and let it be μ_j.
We know that if μ_j is inconsistent then so is μ_i, from Lemma 7.
So, if μ_i is consistent then so is μ_j. And this is so for every μ_j.
So Ω_y is consistent.
So, Π_n is consistent.

(11) (**Decomp** $\Box$)
For some $\mu_i \in \Omega_x \in \Pi_{n-1}$ and $\{\Box B\} \subseteq \mu_i$:
Decomp'$(\{\Box B\}, \mu_i, \Omega_x, \Pi_{n-1}) = \Pi_n$, and
$\Pi_n = (\Pi_n - \{\Omega_x\}) \cup \{\Omega_y$ where
$\Omega_y =$
$(\Omega_x - \{\mu_i\}) \cup \{(\mu_i - \{\Box B\}) \cup \{B\}\}$
The only difference between Ω_x and Ω_y is that
μ_i has been replaced by $\mu_i - \{\Box B\}) \cup \{B\}$
Since μ_i is consistent and contains $\Box B$ it follows from Lemma 5
that $\mu_i \cup \{B\}$ is consistent.
Since $\mu_i \cup \{B\}$ is consistent, $(\mu_i \cup \{B\}) - \{\Box B\}$ is also.
So Ω_y is consistent.
So, Π_n is consistent.

SECOND:

The relationship between Π_{n-1} and Π_n is such that every generated model–set in Ω_n has at least one less connective than the number of connectives in the set of which the generating set of formulas is a sub–set. This follows by simply inspecting all clauses of **Decomp** after the first. Since Π_0 is a finite set of formulas, the sequence must terminate.

So the Lemma is proved.

3.3 Completeness

Theorem: *If A is Valid, then A is a theorem of* S4.

This follows from Lemma 9 in the usual way.

□

4. Soundness

It remains to establish that **S4** is sound with respect to the decompositional system. This is done by the usual method of showing that all the axioms are Valid and that the rules of inference preserve validity.

First we need some results about Model–sequences.

Lemma 10: *If* $<\{\{\{\sim A\}\}\}, \ldots >$ *is an inconsistent Model–sequence, then* $<\{\{\{A\}\}\}, \ldots >$ *is a consistent Model–sequence.*

The proof follows from completeness and the negation consistency of **S4**

Lemma 11: *If A is Valid and* $<\{\{\{B\}\}\}, \ldots >$ *is a consistent Model–sequence, then there is a consistent Model–sequence:*

$$<\{\{\{A, B\}\}\}, \ldots >$$

In truth–tree terms, this is the situation when we add to the bottom of one of the open paths of the tree for B the tree for a Valid formula, A, (not the negation of the Valid formula). At least one path will remain open.

Theorem: *If A is a theorem of* **S4**, *then A is Valid.*

Proof:

We show first that all Axioms are Valid, and then that both Rules of inference preserve Validity.

> We have already seen that Axiom *5* is Valid. The same will be the case for the rest. The tests are left to the reader.

We turn to the Rules of Inference.

> **Modus Ponens**
>
> *R1* is *Modus Ponens*. We assume that the premises are Valid and show the conclusion to be Valid. If the premises, A and $(A \rightarrow B)$, are Valid then any Model–sequence which begins with $\{\{\{\sim A\}\}\}$ will be an inconsistent Model–sequence and any Model–sequence which begins with $\{\{\{\sim(A \rightarrow B)\}\}\}$ will be an inconsistent Model–sequence.
>
> For *reductio* we assume that A and $(A \rightarrow B)$, are Valid, and that B is not Valid.
>
> So, there is a consistent Model–sequence with the first element as follows:
>
> $$<\{\{\{\sim B\}\}\}, \ldots >$$
>
> So,
>
> $$<\{\{\{A, \sim B\}\}\}, \ldots >$$
>
> is a consistent Model–sequence, by **Lemma 11.**

But, since $(A \rightarrow B)$ is Valid there is an inconsistent Model–sequence with the first two elements as follows:

$$<\{\{\{\sim(A \rightarrow B)\}\}\}, \{\{\{A, \sim B\}\}\}, \ldots >$$

It follows that

$$<\{\{\{A, \sim B\}\}\}, \ldots >$$

is an inconsistent Model–sequence, contrary to hypothesis.

Necessitation

R2 is *Necessitation.* We assume that the premise is Valid and show the conclusion to be Valid. Now, if the premise, A, is Valid then every Model–sequence which begins with $\{\{\{\sim A\}\}\}$ will be an inconsistent Model–sequence. Let:

$$\Pi_2 = \{\{\{\sim A\}\}\}$$

Also, let:

$$\Pi_1 = \{\{\{\Diamond \sim A\}\}\}$$

and:

$$\Pi_0 = \{\{\{\sim \Box A\}\}\}$$

Since the relationship of Π_0 to Π_1 to Π_2 is in accordance with **Decomp**, and since any Model–sequence:

$$<\Pi_2, \ldots >$$

will be inconsistent, it follows that:

$$<\Pi_0, \Pi_1, \Pi_2, \ldots >$$

is an inconsistent Model–sequence, and that $\Box A$ is Valid.

So **S4** is sound with respect to the decompositional semantics.

□

REFERENCES

Fitting, M.C. 1988. "First-Order Modal Tableux", *Journal of Automated Reasoning*, 1, 191–214.

Hintikka, J.K. 1969. *Models for Modalities*, Reidel, Dordrecht.

Hughes, G.E. and Cresswell, M.J. 1972. *An Introduction to Modal Logic*, Methuen, London, Corrected Repr.

Hunter, G.B.B. 1971. *Metalogic: An Introduction to the Metatheory of Standard First–Order Logic*, Macmillan, London.

Lemmon, E.J. 1966. "Algebraic Semantics for Modal Logics I, II", *Journal of Symbolic Logic*, 31, 46–65, 191–218

McKinsey, J.C.C. 1941. "A solution of the decision problem for the Lewis systems **S2** and **S4** with an application to topology", *Journal of Symbolic Logic*, 6, 117–134.

Semi-analytic Tableaux For Propositional Modal Logics of Nonmonotonicity

Rajeev Goré *
Department of Computer Science
University of Manchester
Manchester, M13 9PL
England

E-mail: rpg@cs.man.ac.uk

Abstract

The propositional monotonic modal logics **K45**, **K45D**, **S4.2**, **S4R** and **S4F** elegantly capture the semantics of many current *nonmonotonic* formalisms as long as (strong) deducibility of A from a theory Γ, $\Gamma \vdash A$, allows the use of necessitation on the members of Γ. This is usually forbidden in modal logic where Γ is required to be empty, resulting in a weaker notion of deducibility.

Recently, Marek, Schwarz and Truszczyński have given algorithms to compute the stable expansions of a *finite* theory Γ in various such nonmonotonic formalisms. Their algorithms assume the existence of procedures for deciding (strong) deducibility in these monotonic modal logics and consequently such decision procedures are important for automating nonmonotonic deduction.

We first give a sound, (weakly) complete and cut-free, semi-analytic tableau calculus for monotonic **S4R**, thus extending the cut elimination results of Schwarz for monotonic **K45** and **K45D**. We then give sound and complete semi-analytic tableau calculi for monotonic **K45**, **K45D**, **S4.2** and **S4F** by adding an (analytic) cut rule. The proofs of tableau completeness yield a deterministic satisfiability test to determine theoremhood (weak deducibility), $\vdash_L A$, because all proofs are constructive. The techniques are due to Hintikka and Rautenberg. We then show that the tableau calculi extend trivially to handle (strong) deducibility, $\Gamma \vdash A$, for *finite* Γ.

Keywords: modal theorem proving, semi-analytic tableaux, nonmonotonic modal logic.

1 Introduction

Propositional modal logics have been used to model epistemic notions like knowledge and belief for quite a while now where the formula $\Box A$ is read as "A is believed" or as "A is known". Given a (monotonic) modal logic S and a set of formulae Γ, the formula A is a monotonic consequence of Γ in S if it is deducible in S from Γ, usually written as $\Gamma \vdash_S A$. The set Γ is usually called a theory and the monotonic consequences of Γ in S are all the formulae deducible from Γ in

*Research supported by the U.K. Science and Engineering Research Council under grant number GR/H/18449.

S; that is $Cn_S(\Gamma) = \{A \mid \Gamma \vdash_S A\}$. The system is "monotonic" in that if A is in $Cn_S(\Gamma)$ then it will be in $Cn_S(\Gamma')$ for any superset Γ' of Γ.

To obtain nonmonotonicity we assume $\neg\Box A$ ("A is not known") if there is no deduction of A in S from Γ and previous assumptions. More formally, the theory T is an S-expansion of theory Γ if it satisfies the equation $T = Cn_S(\Gamma \cup \{\neg\Box A \mid A \notin T\})$. Since T appears in the right hand side, the definition is circular, and consequently, a theory Γ may have zero, one, or more S-expansions. To compensate for this phenomenon, the set of *nonmonotonic* consequences of Γ in S is usually defined as the intersection of all S-expansions of Γ. The new system is "nonmonotonic" because, although A may be a nonmonotonic consequence of Γ, it may not be a nonmonotonic consequence of a superset of Γ.

Concurrently, various nonmodal formalisms have also been used to model epistemic notions giving rise to default logics and autoepistemic logics. Recently, the nonmonotonic modal logics based on the (monotonic) modal logics **K45**, **K45D**, **S4R**, **S4.2** and **S4F** have been shown to capture the minimal model semantics for some of these *nonmodal* nonmonotonic formalisms; for example, nonmonotonic **K45D** captures the semantics of autoepistemic logic while nonmonotonic **S4F** "naturally generalises default logic and autoepistemic logic" [Sch, Trua, Trub, ST]. Indeed, Marek, Schwarz and Truszczyński [MST91] give algorithms to compute the S-expansions of a *finite* theory Γ in a wide class of nonmonotonic modal logics. However their algorithms assume the existence of effective procedures for deciding deducibility, $\Gamma \vdash_S A$, in the underlying (monotonic) modal logic S. Therefore decision procedures for deducibility in these particular (monotonic) modal logics are crucial for automating nonmonotonic deduction in both *modal and nonmodal* formulations.

Using the technique of semi-analytic tableaux we provide deterministic and nondeterministic decision procedures for deducibility in monotonic **K45**, **K45D**, **S4R**, **S4.2** and **S4F**. Specifically, we give sound and complete (nondeterministic) tableau systems for deciding *theoremhood* in each of these (monotonic) modal logics. Each proof of tableau completeness is constructive thereby yielding a *deterministic* test for satisfiability and hence a *deterministic* test for theoremhood. The system for **S4R** does not have the subformula property but is cut-free proving that Gentzen's cut-elimination theorem holds for **S4R**. The systems for **S4.2** and **S4F** also break the subformula property and require an analytic cut rule for completeness but remain decidable. Cut-free systems for monotonic **K45** and **K45D** have been given already by Schwarz [Shv89] in sequent form, but by adding an analytic cut rule we obtain greatly simplified completeness proofs. The techniques are due to Hintikka and Rautenberg [Rau83, Rau85].

In (monotonic) modal logic, the ability to decide *theoremhood* in a logic **L** *does not* automatically enable us to decide (strong) deducibility, $\Gamma \vdash_L A$, in **L** because theoremhood is defined only when Γ is empty. The tableau calculi can be extended to handle deducibility, in a trivial way, as long as Γ is finite.

2 Definitions and Notational Conventions

2.1 Propositional Normal Modal Logics

We consider only propostional modal logics. We use a denumerable set of primitive propositions $\mathcal{P} = \{p_1, p_2, \cdots\}$ and use $\wedge, \neg$ and $\Box$ as primitives. Then the other usual connectives are defined as abbreviations: $(A \vee B) = (\neg(\neg A \wedge \neg B))$; $(A \Rightarrow B) = (\neg(A \wedge \neg B))$; and $(\Diamond A) = (\neg\Box\neg A)$ where the $=$ sign is merely a meta-linguistic notation. We also use 0 to denote a constant false proposition and use $\emptyset$ to denote the empty set.

The definition of (well formed) formulae is as usual. Lower case letters like p and q denote members of $\mathcal{P}$. Upper case letters from the beginning of the alphabet like A and B together with P and Q (all possibly annotated) denote formulae. Upper case letters from the end of the alphabet like X, Y, Z (possibly annotated) denote *finite* (possibly empty) sets of formulae.

The logics we consider are all normal extensions of the minimal normal modal logic **K** and are axiomatised by taking the rules of *necessitation and modus ponens as inference* rules, by taking the axiom schemas of classical propositional logic, and by taking K and combinations of formulae from Figure 2.1 as further axiom schemas. For an introduction to modal logics see [HC84].

The name of a logic is usually formed by concatenating the names of its (modal) axiom schemas to **K** to denote that the logic is a normal extension of **K**. However we use the traditional names of the more famous logics; for example **S4** is **KT4** and **S4.2** is **KT42**. The logic **S4R** is also known in the literature as **S4.4** and as **SW5**, while **S4F** is also known as **S4.3.2** [Seg71].

2.2 Deducibility and Theoremhood

A deduction of a formula A in logic **L** from a finite set of formulae Γ is a finite sequence of formulae $A_1, A_2, \cdots, A_n$ such that $A_n = A$ and each A_i is: (1) a member of Γ; or (2) an instance of an axiom schema of **L**; or (3) equal to $\Box A_j$ for some $j < i$; or (4) obtained from some A_j and A_k via modus ponens where $k < i$ and $j < i$. We write $\Gamma \vdash_L A$ to indicate there is a deduction of A in **L** from Γ.

This notion of deduction is stronger than the one usually employed in modal logic where "$\Gamma \vdash_L A$" corresponds to the statement that for some finite subset $\{A_1, A_2, \cdots, A_n\}$ of Γ, we have $\vdash_L A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow A$ [Gol87]. If Γ is empty then the two notions coincide and we say that A is a **theorem** of **L** if $\vdash_L A$. For example, $p \vdash_L \Box p$ is a perfectly legal deduction in our formulation but $\vdash_L p \Rightarrow \Box p$ is rarely a theorem in modal logics. Clearly, the deduction theorem is the key but for most of this paper we shall deal with the weaker notion called theoremhood.

2.3 Kripke Semantics

We assume familiarity with the notions of Kripke frames $\langle W, R \rangle$ and Kripke models $\langle W, R, V \rangle$. A possible world w satisfies an atomic formula p *if and only if* $w \in V(p)$. We write this as $w \models p$ and write $w \not\models p$ to mean "not $w \models p$". The semantics of the other connectives and modal operators are as usual.

We often use annotated names like w_1 and w_2 to denote possible worlds. Unless stated explicitly, there is no reason why w_1 and w_2 cannot name the same world. We use R as a name of an axiom schema and also for the reachability relation but the context should make clear which R is meant.

In any model $\langle W, R, V \rangle$, a formula A is **true in a world** $w \in W$ if $w \models A$. A formula A is **valid in a model** $\mathcal{M} = \langle W, R, V \rangle$, written as $\mathcal{M} \models A$, if it is true in all worlds in that model. A formula A is **valid in a frame** $\mathcal{F} = \langle W, R \rangle$, written as $\mathcal{F} \models A$, if A is valid in all models based on $\mathcal{F}$. Suppose $\mathcal{C}$ is a class of models, or of frames. A formula A is **valid in a class** $\mathcal{C}$, written as $\mathcal{C} \models A$, if it is valid in every member of $\mathcal{C}$. An axiom is said to be valid in a model (valid in a frame) if all instances of that axiom have that property. If we have a set of formulae X then $\mathcal{M} \models X$ ($\mathcal{F} \models X$) denotes that all members of X are valid in $\mathcal{M}$ ($\mathcal{F}$).

Let $\mathcal{C}$ be either a collection of models, or of frames. Then logic **L** is **sound with respect to** $\mathcal{C}$ if for every formula A we have that $\vdash_{\mathbf{L}} A$ implies $\mathcal{C} \models A$ [HC84]. Logic **L** is **complete with respect to** $\mathcal{C}$ if for every formula A we have that $\mathcal{C} \models A$ implies $\vdash_{\mathbf{L}} A$ [HC84]. A logic **L** is **determined** or **characterised** by a class $\mathcal{C}$ if it is both

sound and complete with respect to C; that is, when $C \models A$ iff $\vdash_{\mathbf{L}} A$.

A frame $\langle W, R\rangle$ is: **reflexive** if $\forall w \in W, wRw$; **transitive** if $\forall w_1, w_2, w_3 \in W, w_1Rw_2$ and w_2Rw_3 implies w_1Rw_3; **serial** if $\forall w_1 \in W, \exists w_2 \in W, w_1Rw_2$; **symmetric** if $\forall w_1, w_2 \in W, w_1Rw_2$ implies w_2Rw_1; and **convergent** if $\forall w_1, w_2 \in W, \exists w_3 \in W, w_1Rw_3$ and w_2Rw_3.

If $\langle W, R\rangle$ is a frame where R is transitive, then a **cluster** C is a maximal subset of W such that for all *distinct* worlds w and w' in C we have wRw' and $w'Rw$. A cluster is **degenerate** if it is a single irreflexive world, otherwise it is **nondegenerate**. A nondegenerate cluster is **proper** if it consists of two or more worlds. A nondegenerate cluster is **simple** if it consists of a single reflexive world. Note that in a nondegenerate cluster, R is reflexive, transitive *and symmetric*. If the frame is transitive and convergent then there is guaranteed to be a **last** cluster which is reachable from every other cluster. In the case where the clusters form a tree (or a linear sequence), the leaf nodes are known as **final clusters**. For an introduction to Kripke frames, Kripke models and the notion of clusters see Hughes and Cresswell [HC84].

A frame is an **L**-frame if it meets the conditions for **L** as shown in Figure 2. It is known that logic **L** is characterised by the class of all **L**-frames [Seg71, pages 77-78 and 160], [Sch]. Then, a model $\mathcal{M} = \langle W, R, V\rangle$ is an **L-model** if $\langle W, R\rangle$ is an **L**-frame. Also, $\mathcal{M}$ is an **L-model for** (a finite set of formulae) X if there exists $w \in W$ such that $w \models X$. Recall that $w \models X$ means that $w \models A$ for all $A \in X$. A formula A is **L-valid** iff A is valid in all **L**-models, and hence in all **L**-frames. A finite set X is **L-satisfiable** iff there exists an **L**-model for X. So, X is **L-unsatisfiable** iff there are no **L**-models for X.

3 Semantic Tableaux and Tableau Rules

Since our tableau systems work with *finite* sets of formulae, we use the following notational conventions: (1) P and Q stand for formulae; (2) U, X, Y, Z stand for finite sets of formulae; (3) "$X; Y$" stands for $X \cup Y$; (4) "$X; P$" stands for $X \cup \{P\}$; (5) $\Box X$ stands for $\{\Box P \mid P \in X\}$ and (6) $\neg\Box X$ stands for $\{\neg\Box P \mid P \in X\}$. To minimise the number of rules, we work with primitive notation in terms of $\neg, \Box$ and $\wedge$. Each of our tableau rules has a dual rule which can be easily obtained by using the definition of $\Diamond$ as $\neg\Box\neg$. The tableau systems and the completeness proofs are based on those of Rautenberg [Rau83].

3.1 Syntax of Tableau Systems

Tableau systems consist of a collection of tableau (inference) rules. A tableau rule consists of a **numerator** above the line and

Axiom Name	Defining Formula	Alternative Names
K	$\Box(A \Rightarrow B) \Rightarrow (\Box A \Rightarrow \Box B)$	
T	$\Box A \Rightarrow A$	M
4	$\Box A \Rightarrow \Box\Box A$	
5	$\Diamond A \Rightarrow \Box\Diamond A$	E
D	$\Box A \Rightarrow \Diamond A$	
R	$\Diamond\Box A \Rightarrow (A \Rightarrow \Box A)$	W5
F	$(\Box A \Rightarrow B) \vee (\Diamond\Box B \Rightarrow A)$	
2	$\Diamond\Box A \Rightarrow \Box\Diamond A$	

Figure 1: Axiom names and associated schemas.

L	L-frame
K45	type 1: a lone cluster (degenerate or nondegenerate); or type 2: a degenerate cluster followed by a nondegenerate cluster
K45D	type 1: a lone nondegenerate cluster; or type 2: a degenerate cluster followed by a nondegenerate cluster
S4R	type 1: a lone nondegenerate cluster; or type 2: a simple cluster followed by a nondegenerate cluster
S4F	a sequence of at most two nondegenerate clusters
S4	any reflexive and transitive frame
S4.2	any reflexive, transitive and convergent frame

Figure 2: Definition of **L**-frames.

a list of **denominators** (below the line). The denominators are separated by vertical bars. The numerator is a finite set of formulae and so is each denominator. We use the terms numerator and denominator rather than premiss and conclusion to avoid confusion with the sequent terminology.

Figure 3 shows the tableau rules we require. Each tableau rule is read downwards as "if the numerator is **L**-satisfiable, then so is one of the denominators". A tableau calculus is a finite collection of tableau rules identified with the set of its rule names. Thus the tableau calculus for propositional classical logic **PC** is $CPC = \{(0), (\neg), (\theta), (\wedge), (\vee)\}$. The other tableau calculi we consider are shown in Figure 4. The $(S4.2)$ rule is the only potentially dangerous rule since its denominator contains a formula to which the rule can be applied in an endless fashion. To forbid this the new formula is marked with a star and the $(S4.2)$ rule is restricted to apply only to non-starred formulae. All other rules must treat starred formula as if they were non-starred.

Following Rautenberg [Rau83, Rau85], a CL-**tableau** for X is a finite tree $\mathcal{T}$ with root X whose nodes carry finite formula sets stepwise constructed by the rules of CL according to: if a rule with n denominators is applied to a node then that node has n successors with the proviso that if a node E carries a set Y and Y has already appeared on the branch from the root to E then E is an end node of $\mathcal{T}$. A tableau is **closed** if all its end nodes carry $\{0\}$. A set X is CL-**consistent** if no closed CL-tableau for X exists.

When formulated using sets rather than multisets, tableau systems include an implicit rule of contraction since the set $X; P; P$ is the same as the set $X; P$. In order to eliminate contraction, we explicitly build contraction into the rules. For example, the (T) rule contains a form of contraction on $\Box P$ since $\Box P$ is carried from the numerator into the denominator. We return to this point later in Section 6.

For a formula A, the *finite* set of all subformulae $Sf(A)$ is defined inductively as usual where $A \in Sf(A)$. For any finite set X :

- let $Sf(X)$ denote the set of all subformulae of all formulae in X;
- let $\neg Sf(X)$ denote $\{\neg P \mid P \in Sf(X)\}$;
- let $\tilde{X}$ denote the set $Sf(X) \cup \neg Sf(X) \cup \{0\}$;
- let $X^*_{CK45} = X^*_{CK45\dagger} = X^*_{CK45D} = X^*_{C45D\dagger} = \tilde{X}$;
- let $X^*_{CS4R} = Sf(\Box\tilde{X})$.
- let $X^*_{CS4.2\dagger} = X^*_{CS4F\dagger} = Sf(\Box\neg\Box\tilde{X})$.

Thus, a tableau system CL has the subformula property if $X^*_{CL} = \tilde{X}$, and when this

$$(\wedge)\ \frac{X; P \wedge Q}{X; P; Q} \qquad (0)\ \frac{P; \neg P}{0} \qquad (\vee)\ \frac{X; \neg(P \wedge Q)}{X; \neg P \mid X; \neg Q}$$

$$(\neg)\ \frac{X; \neg\neg P}{X; P} \qquad (\theta)\ \frac{X; Y}{X} \qquad (T)\ \frac{X; \Box P}{X; \Box P; P}$$

$$(45)\ \frac{\Box X; \neg\Box Y; \neg\Box P}{X; \Box X; \neg\Box Y; \neg\Box P; \neg P} \qquad (S4)\ \frac{\Box X; \neg\Box P}{\Box X; \neg P}$$

$$(45D)\ \frac{\Box X; \neg\Box Y; \neg\Box P}{X; \Box X; \neg\Box Y; \neg\Box P; \neg P}$$ where $Y; P$ may be empty

$$(R)\ \frac{X; \neg\Box P}{X; \neg\Box P; \neg P \mid X; \neg\Box P; \Box\neg\Box P; P}$$

$$(S4F)\ \frac{U; \Box X; \neg\Box P; \neg\Box Y}{U; \Box X; \neg\Box P; \neg\Box Y; \Box\neg\Box P \mid \Box X; \neg\Box P; \neg\Box Y; \neg P}$$

$$(S4.2)\ \frac{X; \neg\Box P}{X; \neg\Box P; \Box\neg\Box P \mid X; \neg\Box P; \Box(\neg\Box\neg\Box P)^*}$$ where $\neg\Box P$ is not starred

(sfc)

$$\frac{X; \neg(P \wedge Q)}{X; \neg P; \neg Q \mid X; \neg P; Q \mid X; P; \neg Q}$$

$$\frac{X; \neg\Box P}{X; \neg\Box P; P \mid X; \neg\Box P; \neg P}$$

$$\frac{X; \Box P}{X; \Box P; P \mid X; \Box P; \neg P}$$

$(sfcT)$

$$\frac{X; \neg(P \wedge Q)}{X; \neg P; \neg Q \mid X; \neg P; Q \mid X; P; \neg Q} \qquad \frac{X; \neg\Box P}{X; \neg\Box P; P \mid X; \neg\Box P; \neg P}$$

Figure 3: Tableau rules.

CL	Static Rules	Transitional Rules	Structural Rules
CPC	$(0),(\neg),(\wedge),(\vee)$	–	(θ)
$CS4$	$(0),(\neg),(\wedge),(\vee),(T)$	$(S4)$	(θ)
$CK45$	$(0),(\neg),(\wedge),(\vee)$	(45)	(θ)
$CK45D$	$(0),(\neg),(\wedge),(\vee)$	$(45D)$	(θ)
$CS4R$	$(0),(\neg),(\wedge),(\vee),(T),(R)$	$(S4)$	(θ)
$CK45\dagger$	$(0),(\neg),(\wedge),(sfc)$	(45)	(θ)
$CK45D\dagger$	$(0),(\neg),(\wedge),(sfc)$	$(45D)$	(θ)
$CS4.2\dagger$	$(0),(\neg),(\wedge),(sfcT),(T),(S4.2)$	$(S4)$	(θ)
$CS4F\dagger$	$(0),(\neg),(\wedge),(sfcT),(T),(S4.2)$	$(S4F),(S4)$	(θ)

Figure 4: Tableau Calculi

holds, CL is said to be **analytic** [Fit83]. If $\tilde{X} \subset X^*_{CL}$ then CL breaks the subformula property and is said to be **semi-analytic** since CL may not be a decision procedure. However, as long as X^*_{CL} is *finite* then any CL-tableau for X is guaranteed to terminate (for finite X) and there are only a finite number of such tableaux so that even a semi-analytic CL *is* a decision procedure. Some of our systems are semi-analytic since they break the subformula property, and some even contain an (analytic) cut rule, but in each case, X^*_{CL} is finite.

Intuitively, we can associate the numerator and the denominator of a tableau rule with possible worlds. We classify a rule as a **static rule** if the numerator and the denominator correspond to the same world and classify a rule as a **transitional rule** if at least one of the denominators corresponds to a different world. Note that there is only one explicit structural rule (θ).

A set X is **closed with respect to a tableau rule** if, whenever (an instantiation of) the numerator of the rule is in X, so is (a corresponding instantiation of) at least one of the denominators of the rule. If $\mathcal{C}$ is a finite collection of tableau rules then a set X is **closed with respect to** $\mathcal{C}$ if it is closed with respect to each rule in $\mathcal{C}$. For each **L**, a set X is CL**-saturated** if it is CL-consistent and closed with respect to the static rules of CL.

Lemma 1 *For each CL-consistent X there is an effective procedure to construct some finite CL-saturated X^* with $X \subseteq X^* \subseteq X^*_{CL}$.*

Such CL-saturated sets are important because they provide a direct connection between the syntactic and semantic aspects of tableau systems. This is the subject of the next section.

3.2 Soundness and Completeness of Modal Tableau Systems

For soundness we have to show that for each $\mathcal{C}$L-tableau rule: if the numerator is **L**-satisfiable then at least one of the denominators is **L**-satisfiable. For completeness we have to show that if there is no closed CL-tableau for X then X has an **L**-model (i.e. there is an **L**-model which is an **L**-model for X). The basic idea is due to Hintikka.

The following definition from Rautenberg [Rau83] is central for the model construction mentioned above. A **model graph** for some finite fixed set of formulae X is a finite **L**-frame $\langle W, R\rangle$ such that all $w \in W$ are CL-saturated sets with $w \subseteq X^*_{CL}$ and

(i) $X \subseteq w_0$ for some $w_0 \in W$;

(ii) if $\neg\Box P \in w$ then there exists some $w' \in W$ with wRw' and $\neg P \in w'$;

(iii) if wRw' and $\Box P \in w$ then $P \in w'$.

Lemma 2 *If $\langle W, R\rangle$ is a model graph for X then there exists an* **L***-model for X at node w_0 [Rau83].*

This model graph construction is similar in spirit to the subordinate frames construction of Hughes and Cresswell [HC84] except that they use maximal consistent sets and do not consider cycles, giving infinite models rather than finite models.

4 Soundness and Completeness

4.1 Soundness

Theorem 1 (soundness) *If* **L** *is one of* **K45, K45D, S4, S4R, S4.2** *and* **S4F** *then the tableau calculi CL and $CL\dagger$ are sound with respect to* **L***-frames.*

Intuitions for the $(S4)$ rule: If $w \models \neg\Box P$ then $w \models \Diamond\neg P$. Hence there is some w' with wRw' such that $w' \models \neg P$. The $\Box$-formulae of w can be carried into w' by transitivity. Hence the numerator of $(S4)$ represents w and the numerator represents w' and $(S4)$ is a transitional rule.

Intuitions for the $(S4.2)$ rule: Suppose $w \models \neg\Box P$. Consider some w' in the last cluster. By the law of excluded middle $w' \models \Box\neg\Box P$ or $w' \models \neg\Box\neg\Box P$. If $w' \models \Box\neg\Box P$ then $w \models \Box\neg\Box P$. Else, if $w' \models \neg\Box\neg\Box P$ then $w \models \Box\neg\Box\neg\Box P$. Since the numerator and both denominators represent the same world $(S4.2)$ is a static rule.

Intuitions for the $(S4F)$ rule: Assume the model has exactly two clusters C_1 and C_2. If $w \models \neg\Box P; \neg\Box Y$ then there is some w' such that wRw' and $w' \models \neg P$. If w' occurs in C_1 then w must also occur in C_1, hence $w' \models \neg\Box P; \neg\Box Y$. Else if w' occurs in C_2 then all worlds in the model must satisfy $\Box\neg\Box P$. In paticular, $w \models \Box\neg\Box P$. The right denominator represents the transition to w' whereas the left denomonator represents w.

4.2 Completeness of the cut-free systems

Theorem 2 (completeness) *If X is a finite set of formulae and X is CL-consistent then there is an* **L***-model for X on a finite* **L***-frame where* **L** $\in$ **{S4, K45, K45D, S4R}**. *See [Gor93].*

4.3 Completeness for Tableau Systems Containing Analytic Cut

The tableau systems for **S4F** and **S4.2** require the analytic cut rule (sfc) of Smullyan for completeness. Adding analytic cut to the systems for **K45** and **K45D** simplifies the completeness proofs and gives a more direct satisfiability test, as shown below.

A set X is **subformula-complete** if $P \in Sf(X)$ implies that either $P \in X$ or $\neg P \in X$. A crucial consequence of adding analytic cut is the following lemma.

Lemma 3 *If X is closed with respect to $\{(0), (\neg), (\wedge), (sfc)\}$ or with respect to $\{(0), (\neg), (\wedge), (sfcT), (T)\}$ then X is subformula-complete.*

Also note that if $w \prec v$ then $v \subseteq \widetilde{w} = w \cup \neg w \cup \{0\}$; that is, the analytic cut rule can introduce new formulae of the form $\neg P$ into v only if $P \in Sf(w)$. This is crucial for the completeness proofs.

Theorem 3 (completeness) *If X is a finite set of formulae and X is $CL\dagger$-consistent then there is an* **L***-model for X on a finite* **L***-frame where* **L** $\in$ **{K45, K45D, S4.2, S4F}**. *See [Gor93].*

5 Decision Procedures

Each CL gives a nondeterministic procedure to decide whether or not some given (finite) formula A is a theorem of logic **L**. To test whether a formula A is **L**-valid, we simply have to run a CL-tableau construction for $X = \{\neg A\}$. Since X^*_{CL} is finite, there are

only a finite number of such CL-tableaux. If one of them is closed then, by soundness, $\{\neg A\}$ has no **L**-models, and hence A is **L**-valid. If none of these CL-tableaux closes then, by completeness, we can construct a finite **L**-model $\langle W, R, \vartheta \rangle$ which satisfies $\neg A$, hence A is not **L**-valid. We already know that the axiomatically formulated logic **L** is characterised by **L**-frames. That is, we know that a formula A is **L**-valid iff $\vdash_L A$. Therefore, each CL is a highly nondeterministic decision procedure for theoremhood in **L**.

Furthermore, since each completeness proof is constructive, each proof gives a *deterministic* procedure to test whether an arbitrary (finite) set of formulae X is **L**-satisfiable. However, it is known that the satisfiability problem is PSPACE-complete for propositional **S4** and NP-complete for propositional **S5** [HM85]. A similar argument shows that the satisfiability problem for **K45**, **K45D**, **S4R** and **S4F** are also NP-complete whilst the satisfiability problem for **S4.2** is PSPACE-complete.

6 Eliminating Thinning and Contraction

In all of our modal tableau systems, the only explicit structural rule is the rule of thinning (θ). The thinning (or weakening) rule introduces a form of nondeterminism where we have to guess which formula to throw away. That is, we have to guess which formulae are really essential to the proof.

The (θ) rule can be eliminated from all our systems by building the effects of (θ) into the transitional rules and simultaneously changing the basic axiomatic tableau rule from (0) to (0′) as shown below

$$(0') \quad \frac{X; P; \neg P}{0}$$

This technique is used by Fitting [Fit83] and also works for all the other modal calculi we have presented. See [Gor93].

Some of the tableau rules we have given are not standard; for example, the (T) rule is usually given as:

$$\frac{X; \Box P}{X; P} \ (T')$$

where $\Box P$ is not carried from the numerator into the denominator [Rau83]. It is well known that the rule of contraction, which is implicit in the set formulation, then becomes *essential* for completeness. It is also well known that although contraction becomes essential, it is required *only* for $\Box$-formulae in most normal modal logics, and on both $\Box$-formulae and $\Diamond$-formulae in some symmetric normal modal logics [Fit88]. We have deliberately built contraction into our rules to highlight this fact. We believe that if we interpret ";" as multiset union, and rework our formulation using multisets instead of sets, then all the proofs will still go through with appropriate modifications. That is, the rule of contraction appears to be *eliminable* from our systems as long as the static rules build in contraction as given by our rules. Unfortunately, the proofs become very messy.

Therefore these systems are amenable to the Prolog implementation technique of Fitting [Fit88].

7 Deducibility and Strong Completeness

The notion of deducibility, $\Gamma \vdash_L A$, which we use is stronger than the usual one in modal logic where the weaker notion of theoremhood, $\vdash_L A$, is standard. The semantic notion of characterisation by a class of frames must be strengthened to mirror this change. A modal logic **L** is **strongly characterised** by a class of frames C if for every set of formulae Γ and every formula A,

$$\Gamma \vdash_L A \text{ iff } \forall V, \text{ if } \langle W, R \rangle \in C \text{ and}$$

$$\langle W, R, V \rangle \models \Gamma \text{ then } \langle W, R, V \rangle \models A.$$

It is known that the weak characterisation results of modal logics can be strengthened to strong characterisation results if the

modal logic **L** is canonical and the class of frames $\mathcal{C}$ contains the canonical frame for **L** [McD82, MST91]; see Hughes and Cresswell [HC84] for the meanings of canonicity and canonical frame. The logics we study are known to be strongly characterised by the class of frames shown in Figure 2 because these logics and the corresponding **L**-frames satisfy these two conditions.

We can extend our tableau systems to decide whether Γ deduces A, that is, if $\Gamma \vdash_L A$. The trick is to start a tableau for $\Gamma \cup \{\neg A\}$ and then to simply add all members of Γ to each new tableau node as it is created. The set Γ now has "global" status, the proofs of soundness and completeness still go through, and the completeness proofs remain constructive. We are now searching for a countermodel for A where every world satisfies Γ. If we find a closed tableau then we know that no such countermodel is possible and hence, by soundness, that any **L**-model validating Γ must validate A; that is, $\Gamma \vdash_L A$. If we find no closed tableau then, by completeness, we can construct an **L**-model validating Γ and containing a world w_0 which falsifies A. Hence $\Gamma \nvdash_L A$.

An easier way to achieve the same effect is to utilise the fact that all of our transitional rules preserve $\Box$-formulae. That is, for $\mathbf{L} \in \{\mathbf{K45}, \mathbf{K45D}\}$ we run a normal tableau for $\Gamma; \Box\Gamma; \neg A$ to decide whether $\Gamma \vdash_L A$. For $\mathbf{L} \in \{\mathbf{S4}, \mathbf{S4R}, \mathbf{S4.2}, \mathbf{S4F}, \}$ we run a normal tableau for $\Box\Gamma; \neg A$. In each case, the members of Γ will be added to each node automatically by CL.

A caveat is in order because our tableaux procedures terminate only because we deal with *finite* sets. Consequently, we cannot handle first order nonmonotonic modal theories where the infinite set Γ contains all ground instances of the formulae in the (real) first order theory Γ'.

8 Sequent Systems

For each tableau rule there is an analogous sequent rule associating the tableau set X with the sequent $\Sigma \longrightarrow \Delta$ by putting $X = (\Sigma \cup \neg\Delta)$ and turning the tableau rule upside down. For example, the sequent analogue of the tableau rule (45) is

$$\frac{\Box\Gamma \longrightarrow \Box\Delta, \Box A, A}{\Box\Gamma \longrightarrow \Box\Delta, \Box A}(\rightarrow \Box : K45)$$

Thus, each tableau system CL has an analogous sequent system $\mathcal{S}L$.

These sequent systems then give a Gentzen characterisation of theoremhood $\vdash_L$ via the equivalence: $\vdash_L A$ iff the sequent $\longrightarrow A$ is provable in $\mathcal{S}L$. In order to handle (strong) deducibility Fitting [Fit83] introduces the notation $\Gamma \Vdash \Sigma \longrightarrow A$ where $\Sigma \longrightarrow A$ takes the role mentioned above and $\Gamma \Vdash \Sigma \longrightarrow A$ takes the role of $\Gamma \vdash_L \widehat{\Sigma} \Rightarrow A$. Note that the sequent arrow changes to classical implication in this transformation and $\widehat{\Sigma}$ denotes the conjunction of all formulae in Σ. We can obtain $\Gamma \vdash_L A$ by putting Σ to be empty.

Some sort of distinction must be made between Γ and Σ because the former are like "global assumptions" while the latter are like "local assumptions" [Fit83]. If we work with transitive logics then we can use the same trick as in our tableau method by asking whether the sequent $\Gamma, \Box\Gamma, \Sigma \longrightarrow A$ is provable, allowing us to drop the distinction between Γ and Σ. This trick will not work for non-transitive logics because the tableau/sequent rules no longer remain sound with the intended semantics.

The fact that our tableau rules extend to handle (strong) deducibility indicates that a form of the deduction theorem goes through because: $\Gamma \vdash_L A$ iff $\vdash_L (\widehat{\Box\Gamma} \wedge \widehat{\Gamma}) \Rightarrow A$ for $\mathbf{L} \in \{\mathbf{K45}, \mathbf{K45D}\}$ and $\Gamma \vdash_L A$ iff $\vdash_L \widehat{\Box\Gamma} \Rightarrow A$ for $\mathbf{L} \in \{\mathbf{S4}, \mathbf{S4R}, \mathbf{S4.2}, \mathbf{S4F}\}$.

9 Related Work

The cut-free tableau systems $CK45$ and $CK45D$ are the tableau versions of Schwarz's sequent systems for **K45** and **K45D** [Shv89]. The systems $CK45\dagger$ and $CK45D\dagger$ are based on the work of Rauten-

berg [Rau83]. The advantage of $CK45\dagger$ and $CK45\dagger$ is that the associated deterministic satisfiability test is purely local. That is, we need only a two level graph representation, and there is no need to check for repeated nodes.

Apparently Serebriannikov has also obtained a sequent system for **S4.2** but I have been unable to trace this work, let alone determine if this system is cut-free. I know of no other systems for **S4F** or **S4.2** although the idea of the ($S4.2$) rule is due to Rautenberg [Rau83].

Zeman [Zem73] gives a tableau system for **S4R** which he calls **S4.4** but Zeman's system is not cut-free, requiring analytic cut. Zeman also gives a sequent system for **S4R** which is an amalgamation of his sequent systems for **S4** and **S5**. Note however that Zeman explicitly uses an **S5** system and his system for **S5** is not cut-free whereas we simply add one extra static rule to $CS4$ to obtain $CS4R$.

Fitting [Fit83] gives strongly complete analytic tableau systems for many modal logics including **S4** but does not give (strongly complete) systems for the logics we have considered. Incidentally, Rautenberg's technique also gives a strong *analytic* tableau system (containing an analytic cut rule) for **S5**.

10 Further Work

Segerberg [Seg71] shows that **S4F** can also be axiomatised as **S4.3** plus the axiom schema $\Diamond(\Box P \wedge \Diamond\Box Q \wedge \neg Q) \Rightarrow P$ where **S4.3** itself is **S4** plus the axiom schema $\Box(\Box P \Rightarrow Q) \vee \Box(\Box Q \Rightarrow P)$. Elsewhere we have given cut-free tableau systems for **S4.3** and other Diodorean modal logics [Gor92]. We believe that a much more elegant system for **S4F** is possible based on the system for **S4.3**.

Our counter-model constructions for $CS4R$ and $CS4F$ are parasitic on the basic $CS4$ counter-model construction because, in both cases, we have to wait until "eventually this process cycles". Are there more direct counter-model constructions for $CS4F\dagger$ like the ones for $CK45\dagger$? Does adding analytic cut to $CS4R$ help ?

11 Conclusions

We have given (semi)analytic tableau systems for the propositional (monotonic) modal logics **K45**, **K45D**, **S4R**, **S4F** and **S4.2**. Each system provides a nondeterministic decision procedure for theoremhood $\vdash_L A$ and each completeness proof gives a determinsitic **L**-satisfiability test. Each tableau system has a sequent analogue giving a Gentzen cut-elimination theorem for **S4R** from the fact that $CS4R$ is cut-free. These systems extend trivially to handle (strong) deducibility, $\Gamma \vdash_L A$, as long as (strong) deducibility allows necessitation on members of the theory Γ. This is usually forbidden in modal logic but is essential for nonmonotonic modal logics.

Recent results indicate that the *nonmonotonic* modal logics based upon **K45**, **K45D**, **S4R**, **S4F** and **S4.2** elegantly generalise various *nonmodal* nonmonotonic formalisms like autoepistemic logics and default logics. Indeed, the currently known algorithms for computing the nonmonotonic consequences of a theory Γ depend on decision procedures for (strong) deducibility in the underlying monotonic modal logic. Therefore our systems are important for automating nonmonotonic deduction in both modal and nonmodal formulations. The limitation is that Γ must be finite, but when this holds, our tableau calculi are directly implementable in Prolog.

Acknowledgements: I am grateful to Harold Simmons for many helpful discussions about strong deducibility, and modal logic in general.

References

[Fit83] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. D. Reidel, Dordrecht, Holland, 1983.

[Fit88] M. Fitting. First order modal tableaux. *Journal of Automated Reasoning*, 4:191–213, 1988.

[Gol87] R. I. Goldblatt. *Logics of Time and Computation.* CSLI Lecture Notes Number 7, CSLI Stanford, 1987.

[Gor92] Rajeev Goré. The cut-elimination theorem for Diodorean modal logics. Technical Report Forthcoming, University of Cambridge, England, 1992. Abstract in the Proc. of the Logic Colloquium, Veszprem, Hungary, 1992.

[Gor93] Rajeev Goré. Semi-analytic tableaux for modal logics of nonmonotonicity. Technical Report forthcoming, University of Manchester, England, 1993.

[HC84] G. E. Hughes and M. J. Cresswell. *A Companion to Modal Logic.* Methuen, London, 1984.

[HM85] J. Y. Halpern and Y. Moses. A guide to the modal logics of knowledge and belief: Preliminary draft. In *Proc. IJCAI*, pages 480–490, 1985.

[McD82] D. McDermott. Nonmonotonic logic II: Nonmonotonic modal theories. *JACM*, 29:33–57, 1982.

[MST91] W. Marek, G. Schwarz, and M. Truszczynski. Modal nonmonotonic logics: ranges, characterisation, computation. Technical Report 187-91, Dept. of Computer Science, University of Kentucky, USA, 1991.

[Rau83] W. Rautenberg. Modal tableau calculi and interpolation. *Journal of Philosophical Logic*, 12:403–423, 1983.

[Rau85] W. Rautenberg. Corrections for modal tableau calculi and interpolation by W. Rautenberg, JPL 12 (1983). *Journal of Philosophical Logic*, 14:229, 1985.

[Sch] Grigory Schwarz. Minimal model semantics for nonmonotonic modal logics. In *Proc. LICS-92.*

[Seg71] Krister Segerberg. An essay in classical modal logic (3 vols.). Technical Report Filosofiska Studier, nr 13, Uppsala Universitet, Uppsala, 1971.

[Shv89] Grigori F. Shvarts. Gentzen style systems for K45 and K45D. In A. R. Meyer and M. A. Taitslin, editors, *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, LNCS 363*, pages 245–256. Springer-Verlag, 1989.

[ST] G. E. Schwarz and M. Truszczynski. Modal logic S4F and the minimal knowledge paradigm. In *Proc. of TARK 1992.*

[Trua] M. Truszczynski. Embedding default logic into modal nonmonotonic logics. In W. Marek, A. Nerode, and V. S. Subramanian, editors, *Proc. of the first international workshop on logic programming and non-monotonic reasoning.*

[Trub] M. Truszczynski. Modal interpretations of default logic. In *IJCAI-91.*

[Zem73] J. J. Zeman. *Modal Logic: The Lewis-Modal Systems.* Oxford University Press, 1973.

Syntax Independent Connections

Jean Goubault
Bull Corporate Research Center, rue Jean Jaurès, Les Clayes sous Bois
LIENS, Ecole Normale Superieure, Paris, France
Jean.Goubault@frcl.bull.fr

Abstract

We present a new approach to classical first-order theorem proving, and derive a method from it, which can handle formulas for which the textual representation is no help in guiding the proof. Inspired by Prawitz's method, it is close in spirit to Bibel's connection method. However, the strategy we use is different: at each stage in the search for a proof, the prover minimizes the branching factor, and thus the loss of information.

We show experimental results on classical test problems, and analyse them. We show that the parameters of the method are measures of the degree of difficulty of the theorem to prove. This leads us to justify future improvements.

1 Introduction

Both the connection method [4] or the method of general matings [2] rely on the textual form of the given formula to guide the search, but what happens when the syntax does not help? Indeed, although the syntax of human-entered formulas is likely to guide the proof effectively, it is not the case for computer-generated ones: proof obligations in specification methods (VDM [15] and Z [24, 25] to name a few), or automatically inferred formulas, representing the semantics of a program to check against a specification, are a case in point.

The aim of this paper is to give this problem — controlling the search in classical first-order theorem proving, without reference to the textual structure of the input formula — a formal treatment. A new automatic proof method, akin to the connection method, is deduced from it, whose source of inspiration goes back to Prawitz's work in the 1960's [20, 21].

The plan of the paper is as follows: we first present Herbrand's theorem under a different lighting (section 2), and deduce a new method aiming at controlling the combinatorial explosion of the search without relying on textual structure (section 3). How to achieve this as best possible, locally, is the question answered in section 4, by minimizing non-determinism, or alternatively, the loss of information, at each step. Optimizations of the basic algorithm are considered in section 5, where redundancy caused by structural symmetries of the problem is minimized, and in section 6, where attempts to test the same solutions twice are discarded. Results on classical test problems are shown and their analysis is pursued in section 7. Section 8 gives some insight on the guts of the method, and shows that its parameters are actually measures of the difficulty of a theorem; moreover, improvements of the method are sketched. We conclude in section 9.

2 Fundamentals

In the framework of classical first-order logic, Herbrand's theorem is usually presented as a semantic result, sometimes called Herbrand-Skolem-Gödel's theorem [10]:

Proposition 1 *A skolemized first-order formula $\forall x_1, \ldots, x_n \cdot M$, where M is quantifier-free, is unsatisfiable if and only if there exists a finite number of closed instances of M whose conjunction is unsatisfiable.*

(assuming the language of M contains a constant)

We can reformulate the theorem by fixing an integer k, serving as a bound on the "finite" occurring in the text of the theorem. The following development is reminiscent of Prawitz's method [20, 21] or the derived V-resolution procedure [7]. We must solve:

INSTANCE: a first-order formula $\forall x_1, \ldots, x_p \cdot M$, and an integer k

QUESTION: is there a k-tuple $(M_1^c, \ldots, M_k^c)$ of closed instances of M such that $M_1^c \wedge \ldots \wedge M_k^c$ is propositionally unsatisfiable?

(the "c" stands for closed)

Take M and make k copies of it, by replacing the free variables of M by new variables; that is, rename each variable x_i into x_i^j, $j = 1, \ldots, k$, in M, yielding the jth copy M_j. Write $M^k = M_1 \wedge \ldots \wedge M_k$, the k-fold copy of M. We shall call k the *copy count*. The problem becomes:

QUESTION: is there a closed substitution σ such that $M^k\sigma$ is propositionally unsatisfiable?

The "closed" requirement is not necessary, so we shall drop it.

A crucial remark is that $M^k\sigma$ is unsatisfiable if and only if σ makes equal (i.e, unifies) enough atomic subformulas of M^k to reduce it to false. Again, this is Prawitz's approach. The question is now:

QUESTION: is there a number v, $2v$ atomic subformulas A_j, A'_j, $j = 1, \ldots, v$, of M^k, and a substitution σ such that:

- (unif) for all $j = 1, \ldots, v$, $A_j\sigma = A'_j\sigma$
- (cover) $(\bigwedge_{j=1}^{v} A_j \Leftrightarrow A'_j) \Rightarrow \neg M^k$ is a (propositional) tautology?

(cover) is a propositional problem, where the propositional variables ξ_j, $j = 1, \ldots, m$ are the atomic subformulas of M^k; from now on, we identify the two notions. Call any finite set E of couples $(\xi_j, \xi_{j'})$ a system of equations. E induces an equivalence relation among variables, defined as the finest that makes equal the ξ_j and the $\xi_{j'}$; we write it E again. E also defines a propositional formula $\bigwedge_{(\xi_j, \xi_{j'}) \in E} \xi_j \Leftrightarrow \xi_{j'}$, which we write E, again. We call E an *equivalence cover* of $\neg M^k$ if $E \Rightarrow \neg M^k$ is a tautology, and we call v the *step count* (this will be justified in section 8).

Let's say that a substitution σ *realizes* a system E of equations between atomic subformulas if for all $(A_j, A_{j'}) \in E$, $A_j\sigma = A_{j'}\sigma$. The previous question is "is there an realizable equivalence cover of $\neg M^k$?"

This suggests a method for proving first-order propositions [11]: fix $k > 0$, enumerate the covers of $\neg M^k$, in ascending v order; if some cover is realizable, then stop: the formula is unsatisfiable; otherwise, there is no proof of the formula in at most k copies. On failure, k may be increased and the whole procedure restarted. Herbrand's theorem guarantees that if the formula is unsatisfiable, such a process will eventually stop.

Experience has shown that there are way too many covers, and that the realizability constraint must be taken into account when producing the covers. We show how this can be done.

3 Principle of the method

Proving can be seen as a game between two players: the first player does the (cover) part, by guessing part or whole of an equivalence cover, in the hope of eventually obtaining a tautology. If he manages to do so, he wins. The second player does the (unif) part, and binds variables to terms to realize the equations submitted by the first player. If this cannot be done, player two wins. The initial formula is unsatisfiable if player one can win the game.

The connection method uses a strategy for player one, coming from Smullyan's semantic tableaux [23]. It consists in identifying *paths* through the formula M^k, which are conjunction of literals. A set of paths is *spanning* if, taken as a disjunction of paths, it is propositionally implied by M^k. The initial formula is unsatisfiable if and only if there is a spanning set of paths and a substitution that unifies two complementary literals in each path. Player one must choose a path, and play some pair of complementary literals on it (a connection). Player two unifies this pair, changes the set of paths accordingly, and the game goes on.

We recast the method by representing propositional formulas as Shannon trees. Take a total order $<$ on propositional variables, then any propositional formula F can be represented as $(\xi \Rightarrow F_+) \wedge (\neg\xi \Rightarrow F_-)$, where ξ does not occur in either F_+ or F_-, and ξ is the smallest variable in F

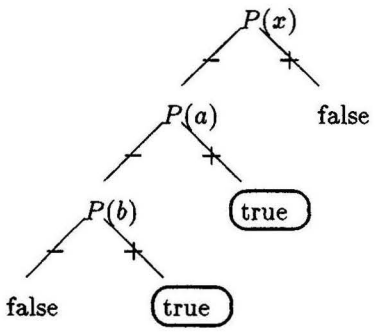
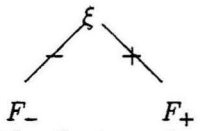


Figure 1: a BDD for $(P(a) \vee P(b)) \wedge \neg P(x)$

(if F has no variable, then it is rewritten into either true or false). For convenience, we adopt C notation, and write it $\xi ? F_+ : F_-$. This makes a tree of F:



where the root node is labelled ξ and has two sons, represented by the trees for F_- and F_+; true and false are the only leaves. Compacting Shannon trees by sharing identical subtrees and eliminating nodes ξ for which F_+ is identical to F_- yields a normal form for boolean formulas, known as binary decision diagrams or BDDs [6], on which boolean functions are fast in practice. BDDs offer an elegant alternative to the boolean simplification tricks in [4]. J.-P. Billon has also used BDDs to ease propositional manipulations in his prover [5], with results similar to ours; however, though BDDs are elegant, they are not indispensable to our method.

The only obstacle for M_k to being an antilogy is the presence of branches that come down from the root and end in a true leave. We call these branches *true branches* (see fig. 2, where true branches are shown as thick lines). A branch is a conjunction of literals: on it, we say that the propositional variable ξ is positive if the branch goes through its right son F_+, and negative if it goes through its left son F_-. A true branch is the conjunction of its positive ξs and of the negations of its negative ξs.

The connection method, transposed to BDDs, finds in each true branch a pair of variables of opposite sign (a *connection*) to unify[1], by testing connections that are highest in the BDD first: this is where branches usually share most. However, the first successful set of connections may reside at the bottom of the BDD, making the problem untractable, whereas another ordering of variables would have made the problem easy.

This is why we need a more clever strategy for player one.

4 Player one's strategy

If the BDD under consideration is not an antilogy, it must contain true branches. The crucial point is that it does not matter which one, because in the end all true branches must contain a pair of unified connections. On a chosen true branch, player one must choose a complementary pair of unifiable atoms. But now the choice matters, for a given choice may prevent him from winning the game. If this happens, he will make another choice of a complementary pair next time he plays, but we would prefer to have the least possible number of choices from start: instead of choosing any true branch, we should ideally choose one that has *the least number of unifiable connections.*

This is a game where the part of a cover that player one can submit must be a single equation. At each point of the game, having chosen a minimal branch, he may have either:

- no move left. Player one would have lost the game anyway; he declares forfeit immediately.

[1] Actually, every true branch is a path, but not all paths need to be true branches.

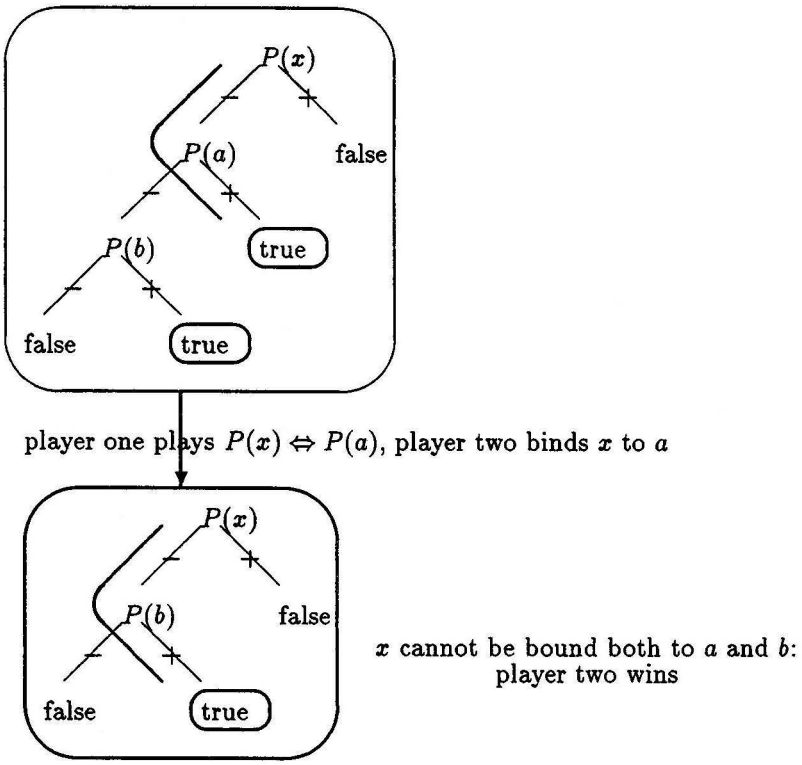


Figure 2: a proof that fails (one copy is not enough here)

- or a number $p \geq 1$ of possible moves, i.e. of connections. The loss of information when choosing one of these moves, if they are assumed equiprobable, is proportional to $\log p$ [22]. Choosing the least p means minimizing the loss of information at each step. In particular, if $p = 1$, this is a deterministic move, with no loss of information.

To compute a minimal branch, we mustn't enumerate all branches, for their number is way too large. Instead, we should be able to compute one in time polynomial in the number of nodes in a BDD, which is usually far less than the number of nodes in the corresponding Shannon tree. Intuitively, a dynamic programming procedure to do this on the BDD F would be to call the function *minbranch*(F) defined with an auxiliary function *minext* as :

1. if F is the leaf "true", fail.
2. if F is the leaf "false", return the empty branch.
3. if $F = A?F_+ : F_-$, compute *minext*(A, F_+) and *minext*($\neg A, F_-$). If both fail, fail; if only one fails, return the other; otherwise, return the one having the least number of unifiable connections.

minext(L, F) fails if *minbranch*(F) fails, otherwise it adds L to the branch *minbranch*(F). The computation of *minbranch* may then be done in a reverse post-order of the BDD, applying it *exactly once to the sub-BDD at each node.*

If n is the number of nodes in the BDD and s is the maximum size of an atomic formula, and using a linear unification algorithm [16], *minbranch* can then be computed in $O(sn^2)$ time. This is improved upon in practice by using a branch-and-bound technique: first one of the subbranches (say, the left one) is computed, then its number of unifiable connections is used as a bound on the computation of the subbranch on the other side (the right one, then).

The problem is that *minbranch* may not compute a minimal branch at all. For example, the formula in figure 4 has as minimal branch $\neg P(a), \neg P(y), P(b)$ (1 unifiable connection), but *minbranch* finds $\neg P(a), P(y), P(x)$ (2 connections). In practice, the number of connections on the

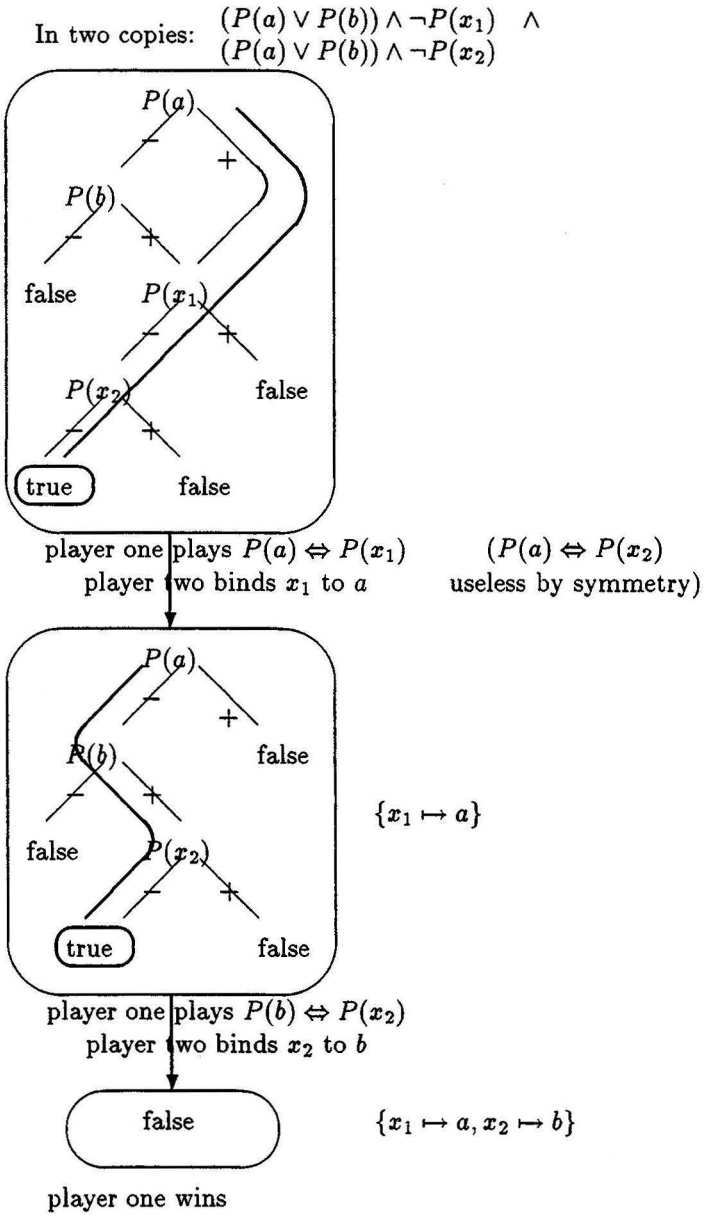


Figure 3: A refutation of $\forall x \cdot (P(a) \vee P(b)) \wedge \neg P(x)$

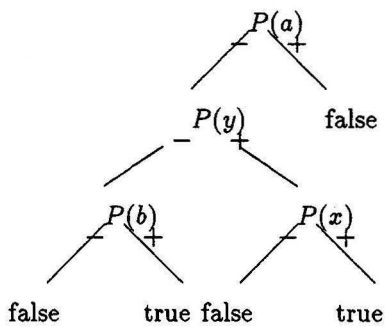


Figure 4: *minbranch* does not always compute a minimal branch

branch returned by *minbranch* is reasonably low. It would be desirable to have a procedure that would really compute a minimal branch; unfortunately, this problem does not seem to be solvable in any time polynomial in n.

5 Handling permutations among copy numbers

More importantly, we must take care of the redundancy of complementary pairs due to copies. At the beginning of the game, for example, we consider $M_1 \wedge \ldots \wedge M_k$; for any permutation σ of $\{1, \ldots, k\}$, this is also $M_{\sigma(1)} \wedge \ldots \wedge M_{\sigma(k)}$. So, whenever in a branch we have a unifiable pair (A_i, A_j) (where the indices i and j indicate to which copy the atoms belong; we call them *copy numbers*), we also have similar branches where we find $(A_{\sigma(i)}, A_{\sigma(j)})$ instead: there are $k!$ of these permutations, though one would suffice.

Modulo a permutation of indices, we can then consider that the couple (A_i, A_j) is actually the couple (A_1, A_2). When player two reduces the BDD by this couple, it also forbids any further permutation of 1 and 2 by player one, though player one can still exchange indices outside $\{1, 2\}$. To implement it, a number u is maintained, such that all indices greater than u can be swapped, but no index less than or equal to u can. By convention, we assume that closed atoms have a copy number of 0; so, initially, $u = 0$. Call a connection *redundant* if player one never needs to play it to win the game. We have:

Lemma 1 *The only non-redundant connections (A_+, A_-) are such that the copy numbers of A_+ and A_- are less than or equal to* $\max(u, 1) + 1$, *and not both are equal to it.*

Proof : Let i_+ be the copy number of A_+, and i_- the copy number of A_-:

- if $i_+ \leq \max(u, 1)$ and $i_- > \max(u, 1)$, then by a permutation of i_- and $\max(u, 1) + 1$, we get indices as in the lemma. The symmetric case obtained by swapping $+$ and $-$ is similar.
- if $i_+ > \max(u, 1)$ and $i_- > \max(u, 1)$, we consider two cases:
  - if after player one plays (A_+, A_-), the BDD is reduced to one having no realizable equivalence cover making equivalent some atoms with copy numbers $i \leq \max(u, 1)$ and $j > \max(u, 1)$: then every realizable cover is the union of two equivalence relations E_1 and E_2, each working on different sets of copy numbers. But then, $(E_1 \wedge E_2) \Rightarrow \neg M^k$ is a tautology if and only if either $E_1 \Rightarrow \neg M_1^k$ or $E_2 \Rightarrow \neg M_2^k$ is a tautology, where M_1^k (resp. M_2^k) is a conjunction of copies of M, precisely the part of M^k that involves only copy numbers in E_1 (resp. E_2). In either case, and because $\max(u, 1) \geq 1$, this provides a proof involving stricly less copy numbers than the original one, so (A_+, A_-) was not necessary.
  - conversely, if such atoms with such i and j exist, then a permutation of j and $\max(u, 1)+1$ yields a couple of unifiable atoms whose copy numbers fit the lemma, on another true branch.

□

Let's see how we use the lemma. In the implementation, copies are not actually built. Instead, the ith copy of an atom A is coded as the couple (A, i) if A has free variables, or $(A, 0)$ if A is closed. Then, the order on copied atoms is done so that if $i < j$, then $(A, i) < (B, j)$ for any A and B (we compare copy numbers first). This way, copying a formula is easy, and yields a BDD of size linear in k. Indeed, we find closed atoms at the top of the BDD, then, going down, we go through the copies numbered 1, 2, ..., as in fig. 5.

The previous lemma justifies the following optimization of the method (k being the number of copies) : the first time player one plays a connection, he must play it between (A, i) and (B, j) with (i, j) being $(0, 1)$, $(1, 1)$ or $(1, 2)$ ($(0, 2)$ is redundant with $(0, 1)$); then for every future step in the search for a proof, u is first set to $\max(i, j, u)$, and player one's connection must then verify $i \leq u$ and $j \leq u + 1$. If we apply this to the example in fig. 5, we indeed find the symmetry argument of fig. 3; be warned, though, that the example is too small to be really representative.

We also fix k in advance, to bound resources from start. In the case where k copies are not enough, we restart the algorithm with a higher k. As there is no proof in k copies, the smallest

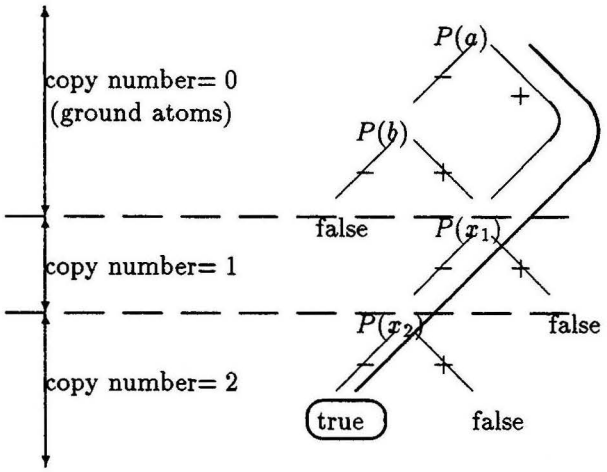


Figure 5: Managing redundancy due to copies

proof must fix at least the first $k+1$ copies by finding connections between all of them. Therefore, while $u \leq k$, player one can be constrained, without loss of completeness, to try only connections between (A, i) and (B, j) with $i \leq u$ and $j = \max(u, 1) + 1$. This way, in only k turns, we get back to a state where only relevant equivalence relations are generated. If we define depth as the copy count, this implements a depth-first iterative deepening search of the space of covers.

Finally, we remark that this handling of permutations is still crude. If we assume that M is the conjunction of n formulas that do not share any first-order variable with the others, hen $M_1 \wedge \ldots \wedge M_k$ is the conjunction, not of k formulas, but really of kn independent formulas C_{ij}, $1 \leq i \leq k$, $1 \leq j \leq n$. These formulas may be swapped independently of each other, so the potential gain of a smart redundancy elimination algorithm is by a factor $(k!)^n$, which is much greater than $k!$. A suitable method for dealing with this remains to be found.

6 Elimination of duplicate covers

Player one tries to build an equivalence relation among atoms one equation at a time. But a sequence of binary equations is not a canonical representation for an equivalence relation. For instance, $(A_0 = A_2, A_0 = A_1)$ and $(A_0 = A_1, A_0 = A_1)$ both define the same equivalence relation. So, some equivalence relations may be tested twice. This may incur a high cost in the traversal of the space of covers :

Lemma 2 *Let $\xi_1, \ldots, \xi_m$ be m variables. Let $\tilde{\omega}_m$ be the number of equivalence relations among the ξ_i. Let X_m be the number of sequences of equations $\xi_{i_k} = \xi_{j_k}$, $k = 1 \ldots p$, with $p < m$, and for every k, $i_k < j_k$ and $\forall k' = 1 \ldots k-1 \cdot (i_k \neq i_{k'} \wedge j_k \neq i_{k'})$. Then:*

$$X_m / \tilde{\omega}_m \simeq C \frac{(m-1)!\sqrt{m/r(m)}}{2^m e^{m/r(m)}}$$

as m tends to $+\infty$, where $C = e\sqrt{2\pi}\, I_1(2\sqrt{2})/(2\sqrt{2}) \approx 32.64$ and $r(m)$ is a slowly growing function of m such that $r(m)e^{r(m)} = m$.

Proof : The $\tilde{\omega}_m$ are known as the Bell numbers [8], and a saddle-point method [26] yields the asymptotic $\tilde{\omega}_m \simeq m! \frac{e^{e^{r(m)}-1}}{\sqrt{2\pi e^{r(m)}}}$, where $r(m)e^{r(m)} = m$ (asymptotically, $r(m) \simeq \log m - \log\log m + \log\log m / \log m + O(\log\log m / \log^2 m)$).

To compute X_m, we choose a couple (ξ_{i_1}, ξ_{j_1}) with $1 \leq i_1 < j_1 \leq m$: this makes $m(m-1)/2$ choices. Then we replace ξ_{i_1} by ξ_{j_1}, and we choose a couple (ξ_{i_2}, ξ_{j_2}) among the remaining $m-1$ variables, and so on. Thus:

$$X_m = 1 + \frac{m(m-1)}{2} + \frac{m(m-1)^2(m-2)}{4} + \ldots + \frac{m!(m-1)!}{1!0!2^{m-1}}$$

$$= \frac{m!(m-1)!}{2^{m-1}} \Sigma_{p=1}^{m} \frac{2^{p-1}}{p!(p-1)!} \simeq \frac{m!(m-1)!}{2^m} \frac{I_1(2\sqrt{2})}{2\sqrt{2}}$$

(for the modified Bessel function I_ν, see [1]) □

In short, the proportion of useful work that is done when sweeping through all sequences of connections is only $O(\frac{2^m e^{m/r(m)}}{(m-1)!\sqrt{m/r(m)}})$, where m is the number of copied atomic formulas. This is roughly inversely proportional to an exponential of $-m$, which decreases rapidly to 0. Therefore, we must avoid this repeated examination of identical equivalence covers.

Our method consists in recording all previously treated equivalence relations in a set: attempts to reexamine equivalence relations are dismissed. Two approaches are then possible. The first one consists in making player one be aware of duplicate equivalence covers. At some step in the search for a proof, the BDD has been reduced by some equivalence cover, call it E. Player one could try to find a branch minimizing the number of unifiable connections that in addition do not extend E to a previously examined equivalence cover. This would however cost a lot of CPU time. So, instead, we use the second approach, where duplicate equivalence covers are not taken into account when looking for a minimal branch, but are simply dismissed, once a branch has been chosen.

To implement the set of equivalence relations, notice that any equivalence relation E can be expressed as a set of rewrite relations $A \rightarrow A'$, where A and A' are equivalent, and $A' < A$ for the order $<$ chosen for the BDD. The sequence of such couples (A, A'), sorted by increasing A' first, then by increasing A, is a normal form for the equivalence relation E. But such sequences are simply words over an alphabet composed of couples, therefore finite sets of equivalence relations correspond to finite languages over this language. Standard techniques can then be used to code a minimal complete automaton recognizing this language, and to update it when adding a word (i.e, an equivalence relation) to the language [14].

This has not been done in the prototype : the set-theoretic primitives of the implementation language were fast enough to code equivalence relations as maps from atoms to their class, and sets as sets.

7 Experimental results

The prototype prover was implemented in a HimML interpreter. HimML [13] is a prototyping language based on Standard ML, extended with fast set-theoretic data types and operations [12]. The various objects (BDDs, branches, covers, etc.) were thus directly coded in terms of objects of the language.

This prototype was applied to Pelletier's problems [17, 18]. The first 17 problems, which are purely propositional are solved as part of the translation process to BDD form; the same can be said of Urquhart's U-problems (pb.71), which we tested up to $n = 165$ (garbage collection times was then becoming much greater than computation time).

The problems 18 through 47, and eight others afterwards, do not involve equality: the minimum copy counts, step counts v and corresponding cardinals of unifiers are shown in fig. 6. We didn't test the problems with equality, assuming they would be too hard to solve without special handling of equality. To explore the search space, a strategy sorting the proposed equivalence relations E by the minimum number $v(E)$ of equations needed to describe them has been used; if an equivalence relation E is considered as a set of non-empty disjoint sets, this number is $\Sigma_{c \in E}(\mathsf{card}\ c - 1)$. The step count is the minimum $v(E)$ for E a cover.

Pelletier's problems are rather small, and our method does not solve all of them. So it does not appear to be quite efficient at first sight. However, this is due to several causes.

First, a naive skolemisation algorithm has been used, which introduces several false dependencies between variables; this can be remedied by using a smart one [3]. Skolemization can have a drastic influence of the speed of a proof method, and we had to use at least two dual tricks : first, existential quantifiers are hoisted out of disjunctions (that is, $(\exists x \cdot P(x)) \vee (\exists y \cdot Q(y))$ is transformed into $\exists x \cdot P(x) \vee Q(x)$), so as to reduce the number of Skolem functions, hence the number of copies needed to match different Skolem functions; second, universal quantifiers are hoisted out of conjunctions. For example, $(\forall x \cdot P(x)) \wedge (\forall y \cdot Q(y))$ is then $\forall x \cdot P(x) \wedge Q(x)$, skolemized as $P(x) \wedge Q(x)$ instead of $P(x) \wedge Q(y)$; this may need twice as many copies, but a good handling of permutations will handle them more efficiently.

monadic logic

pb. number	copies	v	card. unifier
18	2	1	1
19	3	2	2
20	1	1	1
21	2	2	2
22	1	1	1
23	1	2	2
24	1	2	1
25	1	1	1
26	1	8	2
27	2	6	3
28	1	5	2
29	2	10	6
30	1	1	1
31	1	2	1
32	1	3	1
33	1	2	2
34	≥ 3	?	?

full predicate logic

pb. number	copies	v	card. unifier
35	2	1	2
36	3	3	5
37	3	3	5
38	3	25	11
39	1	1	1
40	2	5	3
41	2	2	3
42	2	5	4
43	2	6	6
44	2	3	2
45	3	7	3
46	2	8	3
47	≥ 2	?	?
50	2	2	3
57	1	3	3
59	2	1	1
60	2	4	2
62	2	2	2
66	≥ 3	?	?
67	≥ 3	?	?
68	≥ 3	?	?

Figure 6: Results on Pelletier's problems

Second, our tests of redundancy through permutation of copies is still sloppy. We may only gain a $k!$ factor in speed, instead of a possible $k!^n$. This is in dire need of being improved.

Third, *minbranch* is not guaranteed to find a minimal branch, so that we may not control the non-determinism of the procedure fully. Moreover, it runs in time quadratic in the number of nodes of the BDD. This is not costly if the BDDs remain small, but reducing a BDD by an equation might make it swell by a factor upto 2, so that exponential-sized BDDs might be obtained in the course of the proof. For most problems, BDDs do not swell too much, whatever the order on atoms taken. The notable exception is problem 34 (Andrews' challenge), which was already quite sensitive to order. In fact, the only order that enabled us to translate a Skolem form of problem 34 to a BDD in a reasonable time was based on the left-to-right order of appearance of atomic formulas in the original formula. This order, in general, worked for all other formulas.

8 Some insight, and foreseeable improvements

The interpretation of the step count v is straightforward: it is the least number of non-trivial connections needed to prove the theorem, that is, the least number of unification steps in most proof methods.

The interpretation of the least copy count k is less obvious. A clue is given by the example in figures 1, 2 and 3. If we were to refute $(P(a) \vee P(b)) \wedge \neg P(x)$ by hand, we would just have to let x be a if $P(a)$ is true, and b otherwise. Thus we need to consider two cases; compare this with the fact that the least k for this formula is precisely 2. This is no coicidence: if we try to refute a general formula $\forall x_1, \ldots, x_n \cdot M$, our job is to find a counterexample, i.e. values for the x_i. However, these values may not be expressible as terms in the first-order language at hand. What Herbrand's theorems says then is that these values may be expressed in an extension of the language to *finite-case terms*, of the form $\mathtt{case}\ 1 \mapsto t_1 | \ldots | \mathtt{case}\ k \mapsto t_k$ where the t_js are ordinary terms. The k in these terms is the copy count itself. Hence it is the minimum number of cases to consider when refuting the input formula.

8.1 Splitting

The way we used Herbrand's theorem made us consider a linear arrangement of cases. But humans would consider a hierarchy of cases and sub-cases. To recover this hierarchy, we could use a heuristic that would split the input proposition M into two subproblems M_1 and M_2, such that M is unsatisfiable if and only if both M_1 and M_2 are. A well-known such heuristic is simply called *splitting* and dates back to at least 1969 [21, 7]. In a BDD $A?F_+ : F_-$, splitting means refuting $A \wedge F_+$ and $\neg A \wedge F_-$ separately. A good choice of A might reduce the copy counts of the subproblems to $k/2$: this eventually transforms the original problem into several manageable problems, that is, with copy counts less than or equal to, say, 2 (see fig. 6 for an idea of the right cutoff value).

An example is Pelletier's problem 38, which is an equivalence between two formulas F_1 and F_2. A way of decomposing the problem is by proving two implications. If we do this, one is proved in 2 copies, with a unifier of cardinal 6, and the other is proved in 2 copies, with a unifier of cardinal 4. In general, however, splitting heuristics remain to be tested and evaluated.

8.2 Other improvements

We have already noticed that finding a minimal branch amounts to minimizing the loss of information at each step of the game. On the other hand, reducing a BDD gains information, and the proof is done when we have 100% information. To get there faster, we should start from a state rich in information. Apart from exploiting the textual form of the formula (but our constraint was that it wouldn't help), we could use the fact that most propositions are of the form $A \Rightarrow P$, where A is a conjunction of a priori consistent axioms. This is the heart of set-of-support strategies [7]. Without it, it may be quite difficult to prove Schubert's Steamroller (problem 47): our prototype spent most of its time trying to prove the Steamroller's world consistent.

Another technique is to increase the expressive power of the logic at hand. Known ways range from sorted logics [10] to logics with built-in equational theories (originating with Plotkin's seminal 1972 paper [19]), or with a distinguished equality symbol (which led to several techniques, one of the most promising being proposed in Gallier *et al.*'s 1992 paper [9]).

9 Conclusion and future work

We have presented a new algorithm for proving theorems in classical first-order logic. Though some weaknesses in the implementation prevent it from being as fast as other state-of-the-art systems, is is promising a an automated proof procedure that does not depend on the textual structure of the input formula (it is worthwhile to note that BDDs are not essential to the method, and can be viewed as just another syntactic form). This independence of textual structure is interesting mainly when dealing with computer-generated proof obligations and formulas representing program behavior.

We shall first improve our current implementation and report on the results; the current handling of permutations is much too crude for example. Then, we shall introduce splitting and set-of-support strategies, and evaluate them in the context of our method. Finally, we shall adapt our method to specialized logics, beginning with first-order logic with equality; others will follow. Eventually, we hope that our method will be usable in real applications, in software engineering or elsewhere.

Acknowledgements

I would like the members of IFIP WG 2.3 and the referees for good advice, as well as Joachim Posegga and Dr. Reiner Hähnle for fruitful discussions on several aspects of automatic proof methods.

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* Dover, New York, 1964.

[2] P. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28(2):193–214, 1981.

[3] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Computer Science and Applied Mathematics. Academic Press, 1986.

[4] W. Bibel. *Automated Theorem Proving.* Vieweg, second, revised edition, 1987.

[5] J.-P. Billon. A new approach of theorem proving for non clausal first order logic with equality based on generalized Shannon's decomposition principle. Technical Report RT/91037, Bull S.A. Corporate Research Center, December 1991.

[6] R. E. Bryant. Graph-based algorithms for boolean functions manipulation. *IEEE Transactions on Computers*, C35(8):677–692, august 1986.

[7] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science Classics. Academic Press, 1973.

[8] L. Comtet. *Advanced Combinatorics.* Reidel, Dordrecht, 1974.

[9] J. Gallier, P. Narendran, S. Raatz, and W. Snyder. Theorem proving using equational matings and rigid E-unification. *Journal of the ACM*, 39(2):377–429, April 1992.

[10] J. H. Gallier. *Logic for Computer Science — Foundations of Automatic Theorem Proving.* John Wiley and Sons, 1987.

[11] J. Goubault. A fast method for proving theorems in classical first-order logic. Technical Report RAD/DMA/92018, Bull Corporate Research Center, rue Jean-Jaurès, Les Clayes-sous-Bois, France, July 1992.

[12] J. Goubault. Implementing functional languages with fast equality, sets and maps: an exercise in hash consing. Technical report, Bull S.A. Research Center, rue Jean-Jaurès, 78340 Les Clayes sous Bois, June 1992.

[13] J. Goubault. The HimML reference manual. Technical report, Bull Corporate Research Center, rue Jean Jaurès, 78340 Les Clayes-sous-Bois, France, 1993.

[14] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachussetts, 1979.

[15] C. B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall International Series in Computer Science, 2nd edition, 1990.

[16] M. Paterson and M. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16:158–167, 1978.

[17] F. J. Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[18] F. J. Pelletier. Errata. *Journal of Automated Reasoning*, 4:235–236, 1988.

[19] G. Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[20] D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960.

[21] D. Prawitz. Advances and problems in mechanical proof procedures. *Machine Intelligence*, 4:59–71, 1969.

[22] C. Shannon and W. Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, 1949.

[23] R. M. Smullyan. *First-Order Logic.* Springer Verlag, Berlin, 1968.

[24] J. Spivey. *Understanding Z — A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1988.

[25] J. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International, 1989.

[26] J. Vitter and P. Flajolet. Average-case analysis of algorithms and data structures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 9. Elsevier Science Publishers b.v., 1990.

Tableaux for Reasoning About Objects

P. Gouveia, C. Sernadas, J. Gomes, M.-J. Apolinário

Departamento de Matemática - Instituto Superior Técnico

Av. Rovisco Pais, 1096 Lisboa Codex, PORTUGAL

&

INESC

Apartado 10105, 1017 Lisboa Codex, PORTUGAL

Phone: 351-1-3100322 Fax: 351-1-525843 E-mail: mpg@inesc.pt

Tableau systems for local reasoning about objects are presented. A suitable propositional linear temporal logic for object description is adopted. Our systems were inspired in Fitting's work related to prefixed tableau systems for propositional modal logic but several modifications were made to cope with the specific features of linear temporal logic as well as with objects.

1. Introduction

The object paradigm started with the language SIMULA [Dahl *et al* 67] and is nowadays popular among software engineering researchers namely in the area of systems specification [Loucopoulos&Zicari 92]. An object [SernadasA *et al* 91] includes a set of local attributes and set of local actions (events) and it is able to interact with other objects. Objects are considered suitable abstractions to be used in the different phases of computational systems development being each phase described as a collection (community) of interacting objects. The advantages are the description of static (attributes) and dynamic (events) aspects in an integrated way, the possibility of defining, in each phase, the active and passive entities, modularity [SernadasC *et al* 91a] as well as the possibility of defining systems in layers where a new layer corresponds to introducing more details in a previous one [Ehrich& SernadasA 89,SernadasC *et al* 92a].

Given an object community description (collection of object descriptions and interactions among them) it is important to reason about it, i.e., to prove assertions about the description. These assertions can be local (related with a particular object description) or global (related with the collection of object descriptions and the interactions). This kind of reasoning is important as we can detect inconsistencies, assure that a description really verifies the proprieties we wanted or discover what are the properties implied by the description.

Hence, a logic framework must be adopted. Considering that an object is a dynamic entity that evolves through the occurrence of its events, its life (life cycle, trajectory) can be seen as a sequence of events and each state of the object can be identified with the sequence of events that have occurred so far. An object description will include assertions relating diferent states. So we can adopt a temporal logic to describe objects and, as calculus, a traditional axiomatic perspective [Fiadeiro *et al* 91,SernadasA *et al* 92] or a more operational perspective, using, for instance, tableau systems [Li&SernadasA 91,Gouveia 92]. In what concerns the underlying time structure we can consider a linear one, as we do in this work, or a branching one [Gouveia 92].

In this paper we present tableau systems for local reasoning about objects. It is possible, at local level, to notice two kinds of reasoning: one related to assertions about the values that attributes will have during the object's life and the other related with event occurrence and formulae involving temporal

operators. Here we only are concerned with this second level. So we do not consider attributes and we specify the modifications made by the events through propositional atoms. We adopt a logic based on the usual propositional linear temporal logic [McArthur 76,Emerson 90] including a predicate symbol related with the occurrence of events [SernadasC *et al* 91b,SernadasA *et al* 92] that allows explicit reference to events and their occurrence. The tableau systems presented are based on the prefixed tableau systems for propositional modal logic [Fitting 83]. Using this kind of systems avoids the construction of sets of interdependent tableaux as in other kind of systems [Fitting 83]. For each set of formulae it is only necessary to consider one tableau. From this point of view this work differs from [Li&SernadasA 91]. Moreover, we do not base our tableau construction rules on the axioms of the usual axiomatic systems for temporal logic.

In section 2 we introduce the syntax and semantics of object descriptions. In section 3 we present the syntax and semantics of prefixed formulae, the tableau system, the relevant propositions about the system and several examples. In section 4 we present the conclusions and perspectives of future work.

2. Objects

In this section we present the syntax and semantic of object descriptions. An object description includes a set of events, a set of propositional atoms and a set of formulae of a linear temporal logic. The semantics is based on the usual Kripke semantics being each world a sequence of events.

2.1 Syntax

DEFINITION 2.1: An ***object description signature*** is a pair $\Sigma=(E,P)$ where E is a non empty finite set and P is a finite set of propositional atoms. Elements of E are *events*.

EXAMPLE 2.2: Consider the object description signature $\Sigma_{plane}=(E,P)$ where E={new,close,open,land,take_off} and P={doors_open, on_land}. The description includes events about opening and closing doors, landing and taking off and a propositional atom related with doors and other stating that the plane is on land.

DEFINITION 2.3: Given a object description signature $\Sigma_{ob}=(E,P)$ we define the ***set of formulae*** associated with Σ_{ob}, F_{ob}, as follows: $P \subset F_{ob}$, if $e \in E$ then $OC(e) \in F_{ob}$ and if $f1,f2 \in F_{ob}$ then $\neg f1 \in F_{ob}$, $f1 \wedge f2 \in F_{ob}$, $Ff1 \in F_{ob}$, $Pf1 \in F_{ob}$ and $Xf1 \in F_{ob}$.

We consider as primitives the conectives $\neg$ and $\wedge$, the temporal operators F (sometime in the future), P (sometime in the past) and X (next) and the predicate symbol OC (occurrence) related with the occurrence of events. Informally, OC(e) states that the event e has just occurred. We consider the following abbreviations:

- $f1 \vee f2 =_{def} \neg((\neg f1) \wedge (\neg f2)$
- $f1 \Leftrightarrow f2 =_{def} (f1 \Rightarrow f2) \wedge (f2 \Rightarrow f1)$
- $f1 \Rightarrow f2 =_{def} (\neg f1) \vee f2$
- $Gf =_{def} \neg F \neg f$
- $Hf =_{def} \neg P \neg f$
- $\{f\}e =_{def} XOC(e) \Rightarrow f$.
- $[e]f =_{def} OC(e) \Rightarrow f$

The formula [e]f is a *valuation formula* iff f is propositional formula (i.e. if for every $f' \in F_{ob}$ and $e \in E$ we don't have Ff', Xf', Pf' or OC(e) as subformulae of f).

Informally, a valuation formula states that a formula is true after the occurrence of an event. The formula {f}e is a *safety formula* iff for every $f' \in F_{ob}$ neither Ff' nor Xf' are subformulae of f. Informally, a safety formula imposes some conditions on the occurrence of events: it states that f must be true if e is going to occur.

EXAMPLE 2.4: [new](doors_open∧on_land) and G(OC(take_off)⇒FOC(land)) are formulae of $\in F_{plane}$. The first is a valuation formula.

DEFINITION 2.5 : An ***object description*** is a pair $ob=(\Sigma,F)$ where

- $\Sigma=(\{e_1,...,e_n\}, \{p_1,...,p_m\})$ is an object description signature
- $F \subset F_{ob}$ is finite and $G(OC(e_1)\vee...\vee OC(e_n)) \in F$

The formulae in F are the *axioms* of the object description. The axiom $G(OC(e_1)\vee...\vee OC(e_n))$ reflects the fact that an object is an entity that evolves exclusively through the occurrence of its events.

EXAMPLE 2.6:The pair $plane=(\Sigma_{plane},F)$ is an object description where

$F=$ {G([new](doors_open∧on_land), (a1)
G([close]((¬ doors_open)∧on_land), (a2)
G([open](doors_open∧on_land), (a3)
G([take_off]((¬ doors_open)∧(¬ on_land)), (a4)
G([land]((¬ doors_open)∧on_land), (a5)
X(OC(new))∧GX(¬ OC(new)), (a6)
G({(¬ doors_open)∧on_land}take_off), (a7)
G({doors_open}close), (a8)
G({(¬ doors_open)∧on_land}open), (a9)
G({¬ on_land}land), (a10)
G(OC(take_off)⇒FOC(land)), (a11)
G(OC(new)∨OC(land)∨OC(take_off)∨OC(open)∨OC(close))} (a12).

2.2 Semantics

An interpretation structure for an object description is based on the usual temporal logic semantics. As we assume that an object evolves through the occurrence of its events, at each instant, we can identify the object state with the sequence of events that occurred so far. The sequence can be finite (corresponding to a transient object, i. e., one that after a finite sequence of events is destroyed) or infinite (corresponding to a presistent object, i. e., one that is never destroyed). The empty sequence represents a initial 'pre-natal state'. In what follows we assume, as usual, that, if A is a set then A* is set of all finite sequences of elements of A and, if $w,w_1,w_2 \in A^*$, $|w|$ denotes the length of w and $w_1 \subset w_2$ indicates that w_1 is a proper prefix of w_2.

DEFINITION 2.7: An ***interpretation structure for the object description*** $ob=(\Sigma=(E,P),F)$ is a tuple $M=(W,R,I)$ where

- W is a non empty prefixed closed subset of E^* such that for every $w \in W$ exists at the most one element $w' \in W$ such that $|w'|=|w|+1$
- R is the binary relation on W as follows: $w_1 R w_2$ iff $w_1 \subset w_2$

- $I: W \rightarrow 2^P$ is a map.

The elements of W are the *worlds* (or *states*) of the interpretation structure. The *initial world*, denoted by w_0, is the empty sequence. The *final world*, if it exists, is a world such that $|w| \geq w'$ for every $w' \in W$ and is denoted by w_t. The relation R is the *accessibility relation* of the interpretation structure. From R we can define another binary relation on W, R^1, the *direct accessibility relation*, as follows: $w_1 R^1 w_2$ iff $w_1 R w_2$ and there is no $w_3 \in W$ such that $w_1 R w_3$ and $w_3 R w_2$.

EXAMPLE 2.8 : Consider the object description *plane* presented before and
W={ε, take_off, take_off open} R the accessibility relation on W
$I: W \rightarrow 2^P$ such that $I(\varepsilon)=\emptyset$, I(take_off)=∅, I(take_off open)={doors_open}.
$M=(W,R,I)$ is an interpretation structure for *plane*.

DEFINITION 2.9: Given an interpretation structure $M=(W,R,I)$ for the object description $ob=(\Sigma=(E,P),F)$ and $f \in F_{ob}$ we say that f is ***true at the world*** $w \in W$, denoted by $M,w \models f$ (or $w \models f$ if there is no ambiguity about the interpretation structure) ,iff

- if $f \in P$ then $f \in I(w)$
- if $f=f1 \wedge f2$ then $M,w \models f1$ and $M,w \models f2$
- if $f=\neg f1$ then is not the case that $M,w \models f1$
- if $f=Ff1$ then it exists $w' \in W$ such that wRw' and $M,w' \models f1$
- if $f=Xf1$ then it exists $w' \in W$ such that wR^1w' and $M,w' \models f1$
- if $f=Pf1$ then it exists $w' \in W$ such that $w'Rw$ and $M,w' \models f1$
- if f=OC(e) then w=w'.<e>.

If, in particular, $w=w_0$ then we say that the formula f is *initially true* in M.

DEFINITION 2.10: If *ob* is an object description and $f \in F_{ob}$ then

- f is ***valid***, denoted by $\models f$, iff for every interpretation structure M for *ob* and for every world w in M we have $w \models f$.
- f is ***initially valid***, denoted by $\models_0 f$, iff f is initially true in every interpretation structure M for *ob*.

DEFINITION 2.11: The interpretation structure M for the object description *ob* is a ***model*** for *ob* iff every axiom of *ob* is initially true in M.

So, the axioms of the object description are formulae intended to be true at the initial world and they reflect the properties that should be verified during the object's life. A model represents a possible life of the object, i. e., one that verifies the properties described by the axioms.

EXAMPLE 2.12:Consider the object description *plane* in the example 2.6.:
(1)If axiom (a6), $X(OC(new)) \wedge GX(\neg OC(new))$, is initially true in an interpretation structure for *plane* then the first event to occur is new and this event will never occur after that. This means that we can see new as a *birth* event. Another example is axiom (a9): G({(¬ doors_open)∧on_land}open). If this it is initially

true in an interpretation structure for *plane* then {(¬ doors_open)∧on_land}open will be true at every non initial world which means that during the plane's life the doors will only open provided that they are closed and the plane is on land.
(2) The interpretation structure for *plane* presented in the example 2.8 is not a model for *plane* because, for instance, axioms (a6) and (a9) are not initially valid.

DEFINITION 2.13: An object description is ***inconsistent*** iff there is no model for it.

DEFINITION 2.14: If *ob* is an object description and f∈F_{ob} then a formula f is
- ***ob-valid***, denoted by *ob*⊨f, iff for every model *M* for *ob* and every world w≠w_0 of *M* we have w⊨f
- ***ob-initially valid***, denoted by *ob*$\models_0$f, iff f is initially true on every model for *ob*.

An *ob*-valid formula accounts for a property we want verified on every non initial world of a model, i.e., informally, on every state of the object after its creation during an object's possible life. An *ob*-initially valid formula accounts for a property we want verified on the initial world of every model. By definition, the axioms of the description are *ob*-initially valid formulae.

3. Tableau Systems

In this section we present the tableau system ***OB*** for reasoning about a given object description *ob*=(Σ,*F*). This tableau system was inspired on the prefixed tableau systems for propositional modal logic presented in [Fitting83] but several changes were introduced to cope with the specific features of the underlaying temporal logic. We introduce *prefixed formulae*, i.e., each formula of F_{ob} can be prefixed with a natural number. Intuitively, this prefix represents a world where we want this formula to be true. But we also introduce as formulae, sequences of natural numbers. They represent, syntactically, the worlds and the accessibility relation of the semantic structures. This is an extension of the work presented in [Resende 91] for dynamic logic.

3.1 Syntax and Semantics of Prefixed Formulae

Given a set A: ∘ A={ ∘ a: a∈A}, $\underline{A}$= {$\underline{a}$: a∈A} and $\bar{A}$={$\bar{a}$: a∈A} and the operation ν such that ν(a)=ν($\underline{a}$)=ν($\bar{a}$)=a[#]. $\mathbb{N}$ is the set of natural numbers without 0.

DEFINITION 3.1: The set of all ***prefixed formulae*** associated with the object description *ob*=(Σ,*F*) is F_{obP} ={p:f : p∈*Pr*, f∈F_{ob} }∪*Sq* where *Pr*=$\mathbb{N}$∪ $\underline{\mathbb{N}}$∪$\bar{\mathbb{N}}$ and *Sq*={s_1...s_r∈(*Pr*∪ ∘ *Pr*)*: r>1, s_1∉ ∘ *Pr* }.

Within this context each element of *Pr* is a *prefix*. Elements of $\underline{\mathbb{N}}$ are *initial prefixes* and elements of $\bar{\mathbb{N}}$ are *final prefixes*. The symbol ∘ is the *direct accessibility relation symbol*. The following definitions, related with prefixes, will be necessary when defining the tableau system.

[#]Given an element of A∪$\underline{A}$∪$\bar{A}$ the operation ν returns the underlaying element of A.

DEFINITION 3.2: Let $p,p' \in Pr$, $\circ q \in \circ Pr$, $s=s_1...s_r \in Sq$, $p:f \in F_{ob}P$, $\Pi \subset Pr \cup \circ Pr$ e $\Phi \subset F_{ob}P$

- *pref*(p)=p, *pref*($\circ$ q)=q and *pref*(Π)={*pref*(p'): p'$\in \Pi$}; *pref*(s_i) is a *prefix of* s
- $\nu(s)=\nu(pref(s_1))...\nu(pref(s_r))$
- *comp*(s)={s_1,...,s_r} is the set of the *components of s*; *sucessor*(s_i)=s_{i+1}, $1 \leq i < r$; $\nu(comp(s))=\{\nu(pref(s_i)): 1 \leq i \leq r\}$
- if $\nu(p)=\nu(pref(s_i))$ then the set of *future components of p in s* is $comp_f(s,p)=\{s_j: i<j \leq r\}$; the set of *past components of p in s*, $comp_p(s,p)$, is defined in a similar way
- p' is *accessible from p in s* iff $\nu(p') \in pref(comp_f(\nu(s),\nu(p)))$ or $\nu(p) \in pref(comp_f(\nu(s),\nu(p')))$
- p is *used* in Φ iff $p \in pref(\Phi)=\{p' \in Pr: p':f \in \Phi$ or $p' \in pref(comp(s)), s \in \Phi\}$
- p is *free* in Φ iff p=m$\in \mathbb{N}$, and neither m, $\underline{m}$ nor $\tilde{m}$ are used in Φ.

DEFINITION 3.3: Let $s=s_1...s_r \in Sq$, p,q prefixes, p prefix of s and q free in {s}. The ***future extension of s from p through q***, $ext_f(s,p,q)$, is the set of formulae $s' \in Sq$ such that

- $s'=s_1...s_j\sigma$ where $p=pref(s_j)$ for some $1 \leq j \leq r$ and σ is a permutation of elements of $comp_f(s,p) \cup \{q\}$
- $s' \downarrow comp(s)=s$ where $s' \downarrow comp(s)$ is the restriction of s' to the set *comp*(s)
- in s', $sucessor(q) \notin \circ Pr$.

We define, in a similar way, the ***past extension of s from p through q***, $ext_p(s,p,q)$.

EXAMPLE 3.4: Consider the formula $s=2 \circ 5\ 3$. Then $comp(s)=\{2,3, \circ 5\}$, $pref(\circ 5)=5$, 3 is accessible from 2 in s, $comp_f(s,2)=\{3, \circ 5\}$, $comp_p(s,5)=\{2\}$, $2 \circ 5\ 6\ 3$ is an element of $ext_f(s,5,6)$ but $2\ 6 \circ 5\ 3$ is not an element of $ext_p(s,3,6)$.

DEFINITION 3.5: Let $\Pi \subset Pr$ be a set of prefixes and $M=(W,R,I)$ an interpretation structure for the object description *ob*. We say that it exists an ***interpretation*** I^p of Π on M if

- if Π has no final prefixes then I^p is a map $I^p:\Pi \rightarrow W$ such that $I^p(p)=w_0$, for every initial prefix p and if $\nu(p')=\nu(p'')$ then $I^p(p')=I^p(p'')$
- if Π has final prefixes then W must be finite and I^p is a map $I^p:\Pi \rightarrow W$ verifying the conditions above and $I^p(p)=w_t$, for every final prefix p.

DEFINITION 3.6: A formula $f \in F_{ob}P$ is ***satisfiable*** iff there is an interpretation structure $M=(W,R,I)$ for *ob* such that

- if f=p:f1 then there is an interpretation I^p of {p} on M such that $I^p(p) \models f1$
- if $f=s_1...s_r \in Sq$ then there is an interpretation I^p of *pref*(*comp*(f)) on M such that for every $1 \leq i < r$ $I^p(pref(s_i))RI^p(pref(s_{i+1}))$ and if $s_{i+1} \in \circ Pr$, then $I^p(pref(s_i))R^1I^p(pref(s_{i+1}))$. Notation: $M,I^p \models f$.

DEFINITION 3.7: Let $\Phi \subset F_{ob}P$

- Φ is *satisfiable* iff there is an interpretation structure M and an interpretation I^p of *pref*(Φ) on M such that $M,I^p \models f$ for every $f \in \Phi$
- Φ is *unsatisfiable* iff is not satisfiable.

3.2 Tableaux System *OB*

In this section we present the tableau system ***OB*** for reasoning about the object description $ob=(\Sigma=(E,P),F=\{a1,...,an\})$. Each tableau is a tree such that each node is labelled with a finite subset of F_{obP} and it is built following the ***OB***-rules presented below. A tree t is a *tableau for* Φ iff Φ is the label of the root node of t. If β is a branch then Φ_β denotes the union of the labels of the nodes of β. We will present the ***OB***-rules in the usual way:

$$\frac{\Phi\ ,\ f1,...,fn}{\begin{array}{|c|}\hline \text{rule 1}\\ g1\\ ...\\ gk\\ \hline\end{array}} \qquad \frac{\Phi\ ,\ f1,...,fn}{\begin{array}{|c|c|c|}\hline g11 & ... & gm1\\ ... & & ...\\ g1k_1 & & gmk_m\\ \hline\end{array}\ \text{rule 2}}$$

meaning that if $\Phi\cup\{f1,...,fn\}$ is the union of the labels of all nodes of a tableau branch then we can add to that branch leaf one successor node (rule 1) labelled with $\{g1,...,gk\}$, or m successor nodes (rule 2) with $\{gi1,...,gik_i\}$ labelling node i, $1\le i\le m$. The arity of a rule is the number of successor nodes added. The set $\{f1,...,fn\}$ is the set of *principal formulae* of rules 1 and 2. The set $\{g1,...,gk\}$ is the set of *derived formulae* of rule 1 and $\{gi1,...,gik_i\}$ is the ith set of *derived formulae* of rule 2. We say that a rule *can be applied* to a tableau branch β iff the set of principal formulae of the rule is a subset of Φ_β. Next we present the tableau construction rules, ***OB***-rules, related with the considered primitive conectives and temporal operators and the definition of closed branch and closed tableau.

DEFINITION 3.8: ***OB***-RULES

I. Propositional Rules

$$\frac{\Phi,\ p{:}\neg\neg f}{\begin{array}{|c|}\hline \neg\neg\\ p{:}f\\ \hline\end{array}} \qquad \frac{\Phi,\ p{:}f1\wedge f2}{\begin{array}{|c|}\hline \wedge\\ p{:}f1,\ p{:}f2\\ \hline\end{array}} \qquad \frac{\Phi,\ p{:}\neg(f1\wedge f2)}{\begin{array}{|c|c|}\hline \neg\wedge & \\ p{:}\neg f1 & p{:}\neg f2\\ \hline\end{array}}$$

II. Temporal Rules

$$\frac{\Phi,\ p{:}Xf}{\begin{array}{|c|}\hline Xi\\ p\circ q,\ q{:}f\\ \hline\end{array}} \qquad \frac{\Phi,\ s_1...s_i\circ q...s_r,\ p{:}Xf}{\begin{array}{|c|}\hline Xii\\ q{:}f\\ \hline\end{array}} \qquad \frac{\Phi,\ s_1...s_i\circ q...s_r,\ p{:}\neg Xf}{\begin{array}{|c|}\hline \neg X\\ q{:}\neg f\\ \hline\end{array}}$$

Conditions: q is free on $\Phi\cup\{p{:}Xf\}$, $Sq\cap\Phi=\emptyset$

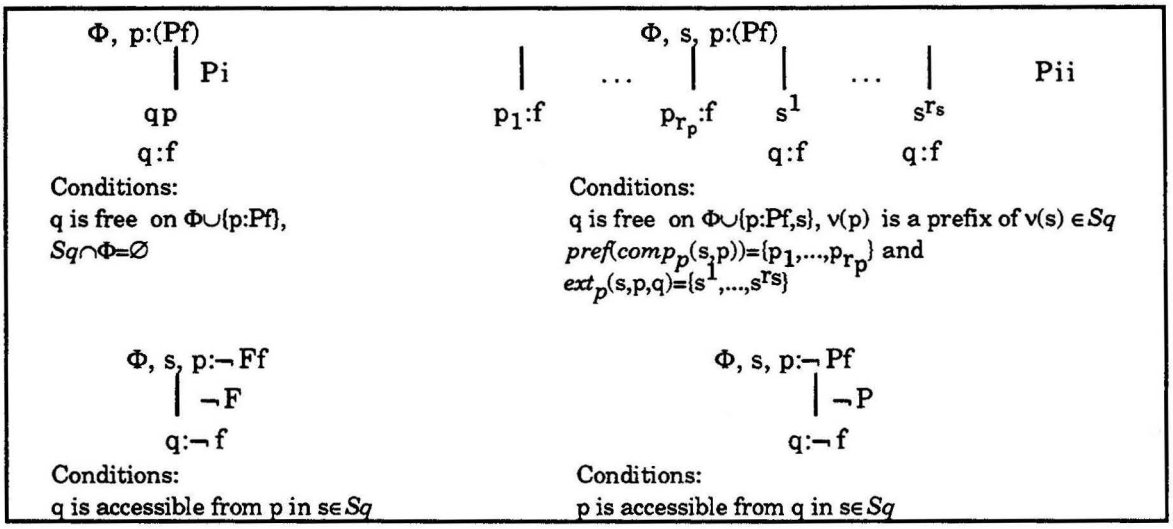
Conditions: $\nu(pref(s_i))=\nu(p)$

Conditions: $\nu(pref(s_i))=\nu(p)$

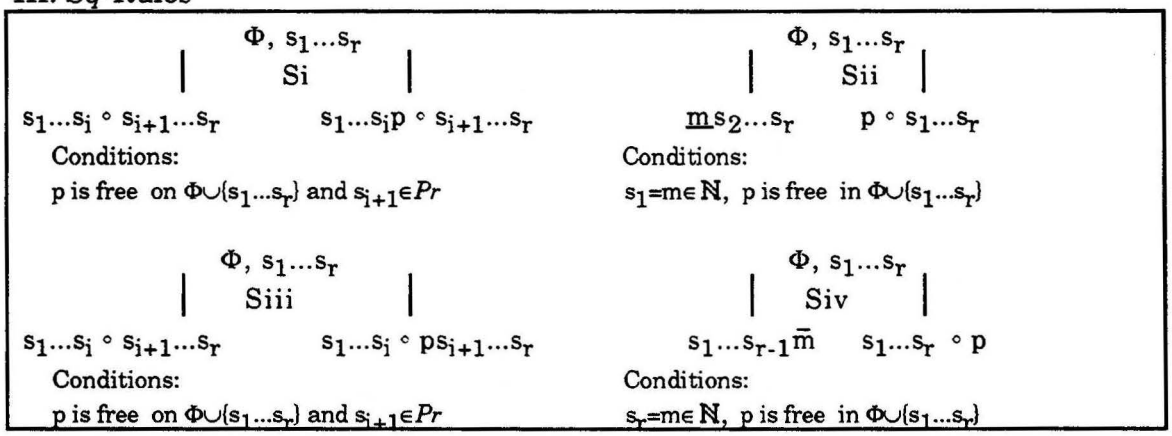
$$\frac{\Phi,\ p{:}(Ff)}{\begin{array}{|c|}\hline Fi\\ p\,q\\ q{:}f\\ \hline\end{array}} \qquad \frac{\Phi,\ s,\ p{:}(Ff)}{\begin{array}{|c|c|c|c|c|}\hline & ... & & ... & \\ p_1{:}f & & p_{r_p}{:}f & s^1 & s^{r_s}\\ & & & q{:}f & q{:}f\\ \hline\end{array}\ Fii}$$

Conditions: q is free on $\Phi\cup\{p{:}Ff\}$, $Sq\cap\Phi=\emptyset$

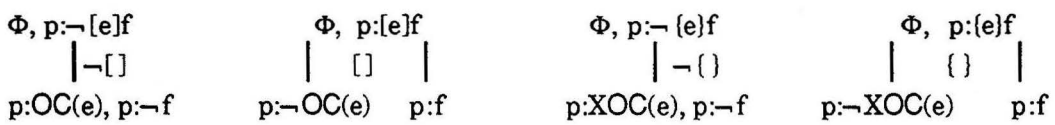
Conditions: q is free on $\Phi\cup\{p{:}Ff,s\}$, $\nu(p)$ is a prefix of $\nu(s)\in Sq$, $pref(comp_f(s,p))=\{p_1,...,p_{r_p}\}$ and $ext_f(s,p,q)=\{s^1,...,s^{r_s}\}$



III. *Sq*-Rules



From the rules presented above we can infer rules related with the abbreviations introduced. We have the usual rules ∨, ¬∨, ⇒, ¬⇒, ⇔ and ¬⇔. The rules ¬Gi, ¬Gii, ¬Hi, ¬Hii, G and H are similar to the rules Fi, Fii, Pi, Pii, ¬F and ¬P, respectively. Finally we have the rules [ ], ¬[ ], { } and ¬{ } presented below.



DEFINITION 3.9: A branch β in a tableau is ***closed*** iff one of the following conditions holds

- {p:f,q:¬f}⊆Φβ and ν(p)=ν(q)
- {p:OC(e1),q:OC(e2)}⊆Φβ and ν(p)=ν(q), e1≠e2
- {p:OC(e)}⊆Φβ or {p:Pf}⊆Φβ and p is an initial prefix or there is an initial prefix q∈*pref*(Φβ) such that ν(p)=ν(q)
- {p:Xf}⊆Φβ or {p:Ff}⊆Φβ and p is a final prefix or there is a final prefix q∈*pref*(Φβ) such that ν(p)=ν(q)
- {s=s1...sr}⊆Φβ and there is si, 1<i≤r, such that *pref*(si) is an initial prefix or there is si, 1≤i<r, such that *pref*(si) is a final prefix.

A ***tableau is closed*** iff all branches are closed.

3.3 Soundness

The tableau system presented above is sound wrt to unsatisfiable finite subsets of F_{obP}, i.e., if there is a closed tableau for Φ then Φ is unsatisfiable. As consequence, if, for instance, $\Phi=\{1{:}\neg f\}$ we can conclude that f is a valid formulae, i.e., f is true at every world of every interpretation structure for *ob*. If $\Phi=\{\underline{m}{:}a1,...,\underline{m}{:}an\}$, being a1,...,an all the axioms of *ob*, then *ob* is an inconsistent object description as there is no interpretation structure such that every axiom is initially true on it, i. e., *ob* as no models. If $\Phi=\{\underline{m}{:}a1,...,\underline{m}{:}an,\underline{m}{:}\neg Gf\}$ then we can conclude that f is *ob*-valid. The following propositions establish these results.

PROPOSITION 3.10: Let β be a branch of a tableau t, r a ***OB***-rule of arity m that can be applied to β and, for $1\leq i\leq m$, $\{gi_1,...,gik_j\}$ the ith set of derived formulae of r. If Φ_β is satisfiable then at least one of the sets $\Phi_\beta\cup\{gi_1,...,gik_j\}$, $1\leq i\leq m$, is satisfiable.

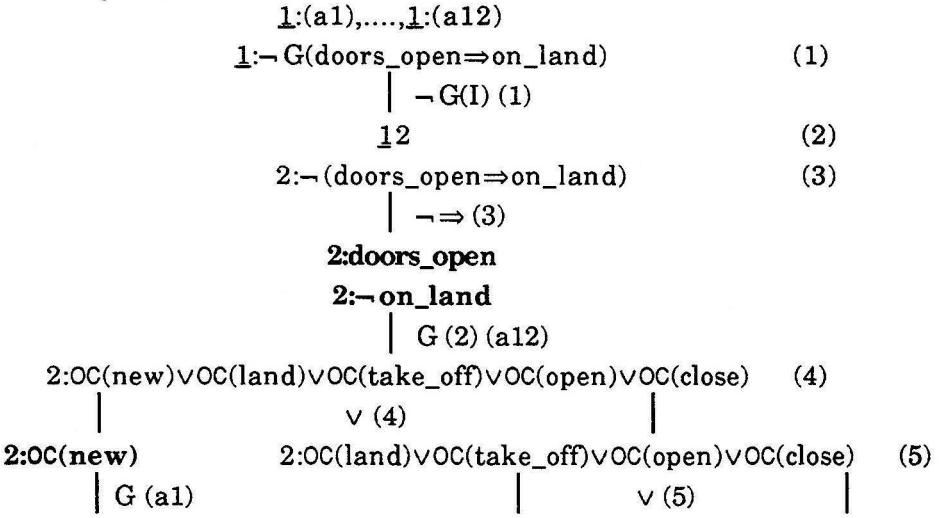
PROPOSITION 3.11: If t is a tableau for Φ and β is a branch of t then: (i) if β is closed then Φ_β is unsatisfiable and (ii) if t is closed then Φ is unsatisfiable.

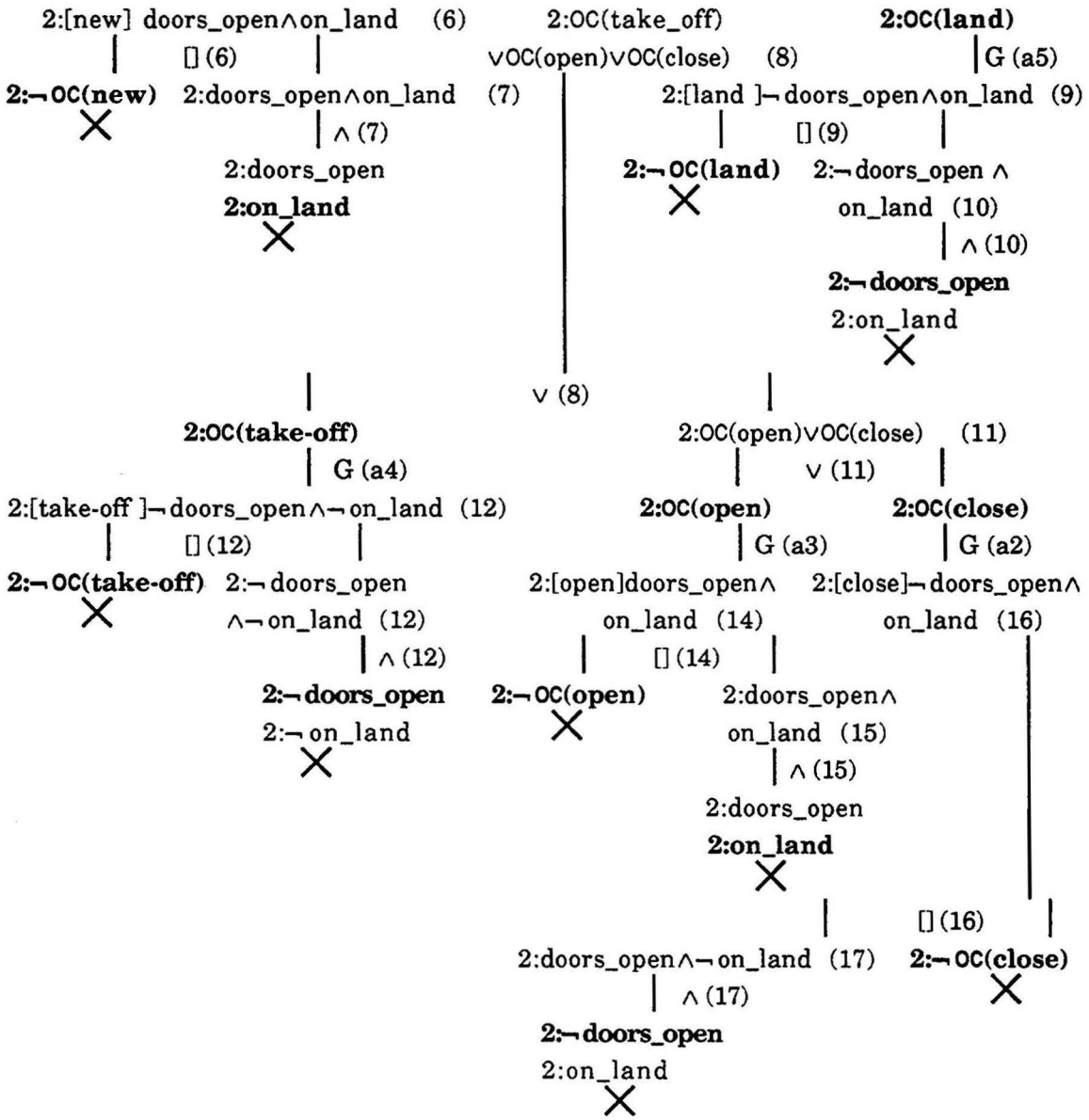
PROPOSITION 3.12: Consider the object description $ob=(\Sigma=(E,P),F=\{a1,...,an\})$ and t a closed tableau for $\Phi\subset F_{obP}$.

(i) if $\Phi=\{p{:}\neg f\}$ then: (a) if p is an initial prefix then f is initially valid and (b) if p is neither initial nor final then f is valid
(ii) if $\Phi=\{\underline{m}{:}a1,...,\underline{m}{:}an\}$ then *ob* is an inconsistent object description
(iii) if $\Phi=\{\underline{m}{:}a1,...,\underline{m}{:}an,\underline{m}{:}\neg Gf\}$ then f is *ob*-valid
(iv) if $\Phi=\{\underline{m}{:}a1,...,\underline{m}{:}an,\underline{m}{:}\neg f\}$ then f is *ob*-initially valid.

Next we present some examples of tableaux within the context of the object description *plane* introduced above. Each closed branch is marked with the symbol × and the formulae that close a branch are printed in bold.

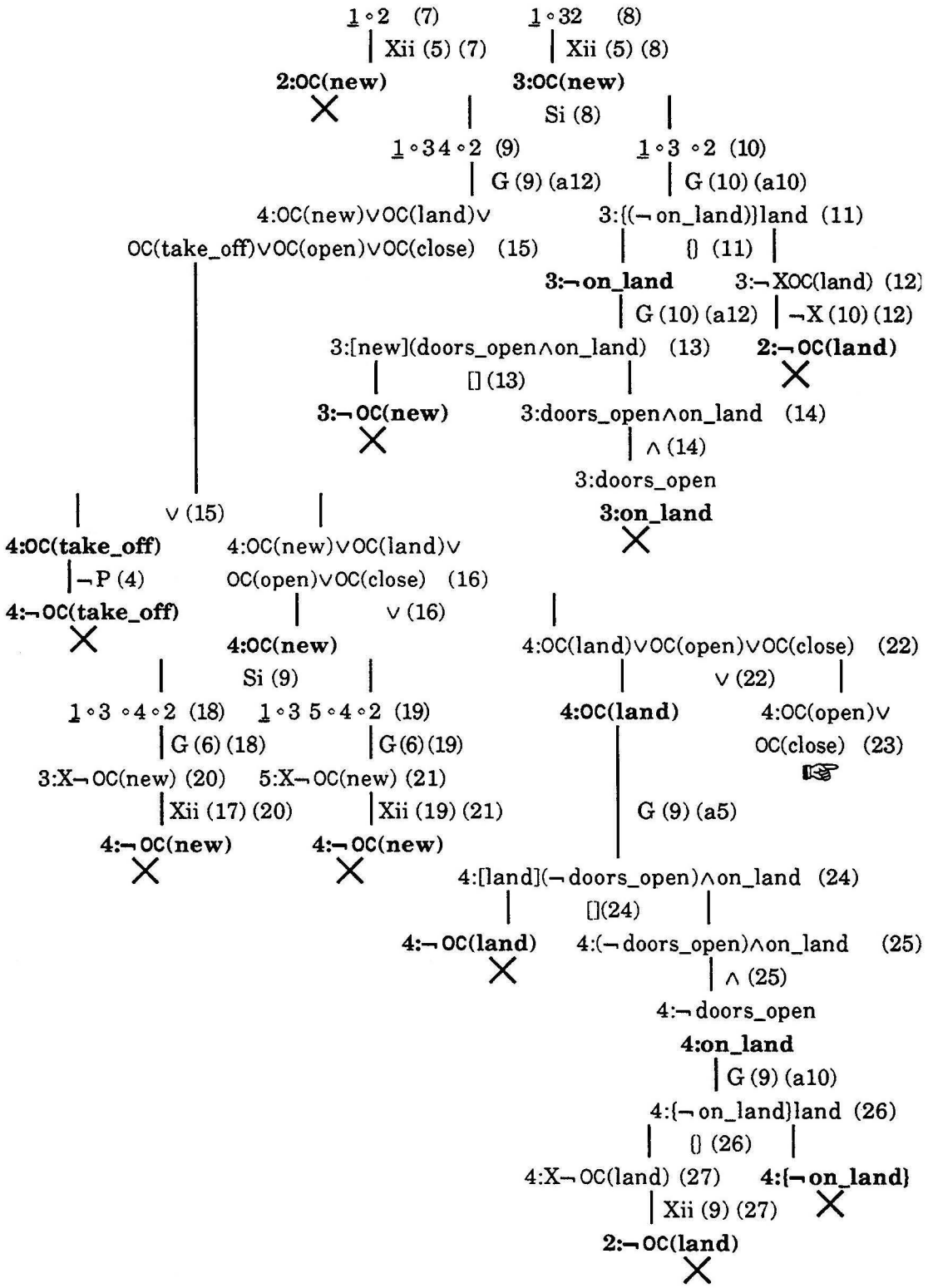
EXAMPLE 3.13:With the tableau system ***PLANE*** we can reason about the object description $plane=(\Sigma_{plane},\{(a1),...,(a12)\})$ given above. Consider the tableau:





As we have a closed tableau for {1:(a1),...,1:(a12),1:¬ G(doors_open⇒on_land)} by proposition 3.12(iii) doors_open⇒on_land is *plane*-valid. So the description *plane* implies that the doors are only open when the plane is on land. The next tableau assures that OC(land)⇒POC(take_off) is also a *plane*-valid formula.

1:(a1),....,1:(a12),
1:¬ (G(OC(land)⇒POC(take_off))) (1)
¬ G(I) (1)
12 (2)
2:¬ (OC(land)⇒POC(take_off)) (3)
¬ ⇒ (3)
2:OC(land)
2:¬ (POC(take_off)) (4)
∧ (a6)
1:XOC(new) (5)
1:GX(¬ OC(new)) (6)
Siii (2)



We can easily see that we could build the branch marked with ☞as we have built the one next to it. For simplicity reasons we don't present here that branch.

4. Conclusions

We presented tableau systems for local reasoning about objects. An object description includes a set of formulae of a propositional linear temporal logic enriched with an event occurrence. Given an object description *ob*, the tableau

sistem ***OB*** associated with it is sound wrt unsatisfiable finite subsets of F_{obP} , i. e., if there a closed tableau for Φ then Φ is an unsatisfiable set. Completness is under current research.

Herein we considered objects within the context of linear temporal logic. An extension of this work for branching temporal logic is presented in [Gouveia92].

As future work we intend to extend the systems presented in order to cope with reasoning about attributes and its values and to cope with global reasoning, i. e., to prove properties related not only with individual object descriptions but also with interaction and reification between objects. The logic we consider suitable for this purpose is presented in [SernadasA *et al* 92] in axiomatic style.

Acknowledgments

This work was partially supported by JNICT Project PMCT/C/TIT/177/90 (OBCALC) and J. Gomes e M.-J. Apolinário have a scholarship from JNICT.

References

[Dahl *et al* 67]: Dahl, O., Myhrhaug, B., Nygaard, K., *Simula 67: Common Base Language*, Norwegian Computing Center, 1967.

[Emerson 90]: Emerson, E., Temporal and Modal Logic, *Handbook of Theoretical Computer Science*, J. van Leewen (ed), Elsevier Science Publishers, 1990, 997-1072.

[Fiadeiro *et al* 91]: Fiadeiro, J., Sernadas, C., Maibaum, T., Saake, G., Proof-theoretic Semantics of Object-oriented Specification Constructs, *Object Oriented Databases: Analysis, Design and Construction*, Meersman, R., Kent, W., Khosla, S. (eds), North Holland, 1991, 243-284.

[Fitting 83]: Fitting, M., *Proof Methods for Modal and Intuitionistic Logics*, Synthese Library 169, Reidel, 1983.

[Gouveia 92]: Gouveia, P., *Tableaux for Local Reasoning About Objects* (in portuguese), MScThesis, IST, 1992.

[Li&SernadasA 91]: Li, R., Sernadas, A., Reasoning About Objects Using Tableau Method, *Journal of Logic and Computation*, 1 (5), North Holland, 1991, 575-611.

[Loucopoulos&Zicari 92]: Loucopoulos, P., Zicari, R. (eds), *Conceptual Modeling Databases and CASE: An Integrated View of Information Systems Development*, John Wiley, 1992.

[McArthur 76]:McArthur, R., *Tense Logic*, Reidel, 1976.

[Resende 91]: Resende, P., *Tableaux for Proposicional Dynamic Logic* (in portuguese), MSc Thesis, IST, 1991.

[SernadasA *et al* 91]: Sernadas, A., Ehrich, H., Sernadas, C., What is an Object, After All?, *Object Oriented Databases: Analysis, Design and Construction*, Meersman, R., Kent, W., Khosla, S. (eds), North Holland, 1991, 39-69.

[SernadasA *et al* 92]: Sernadas, A., Sernadas, C., Costa, F., Object Specification Logic, Research Report DMIST/IST, submited, 1992.

[SernadasC *et al* 91a]: Sernadas, C., Resende, P., Gouveia, P., Sernadas, A., In-the-large Object-oriented Design of Information Systems, *The Object-Oriented Approach in Information Systems*, Van Assche, F., Moulin, B., Rolland, C. (eds), North Holland, 1991, 209-232.

[SernadasC *et al* 92a]: Sernadas, C., Gouveia, P., Gouveia, J., Sernadas, A., Resende, P., The Reification Dimension in Object-oriented Data Base Design, *Specification of Data Base Systems*, Harper, D., Norrie, M. (eds), Springer Verlag, 1992, 275-299.

Semantic Constraint in Model Generation Theorem Proving EXTENDED ABSTRACT

Mark Grundy
Center For Information Science Research,
ANU, ACT, Australia 2600.

1 Introduction

The method of model generation for classical logics [3] may be viewed either as an extension of Smullyan's tableaux, or as a form of hyperresolution. Clauses are entered in the form $A \rightarrow C$, where A is a (possibly empty) set of antecedent formulae and C is a (possibly empty) set of consequent formulae. The initial formulae of tableaux can be represented as $\emptyset \rightarrow C$ (or, simply $\rightarrow C$), tableau extension rules (append rules), are generalised to the form $A \rightarrow C$, where A and C are not empty, and tableau closure rules are generalised to the form $A \rightarrow \emptyset$, or more simply, $A \rightarrow$. The tableaux that result from these systems admit multiple branchings at each node, and multiple means of branch closure, but otherwise act similarly to more traditional classical tableau.

Model Generation Theorem Proving has been realised in a variety of programs, including SATCHMO [6] Ground MGTP (G/MGTP) [4] and Non-Ground MGTP (N/MGTP) [2,5]. G/MGTP is a prover that works on range-restricted problems, in which for each clause, every variable of the consequent appears at least once in the antecedent. G/MGTP can be naturally viewed as a generalisation of ground tableau. In the non-ground form, N/MGTP, this restriction is lifted, but clauses are limited to horn-form. N/MGTP can be seen as a restriction of a more general tableau technique to a resolution-style system.

This work discusses a means for combining these technologies into a system that can significantly reduce the search-space for horn problems.

2 Background and Overview

Refutation procedures such as tableau and resolution depend upon the premise that the negation of a theorem cannot be modelled with the axioms of the logic. Yet

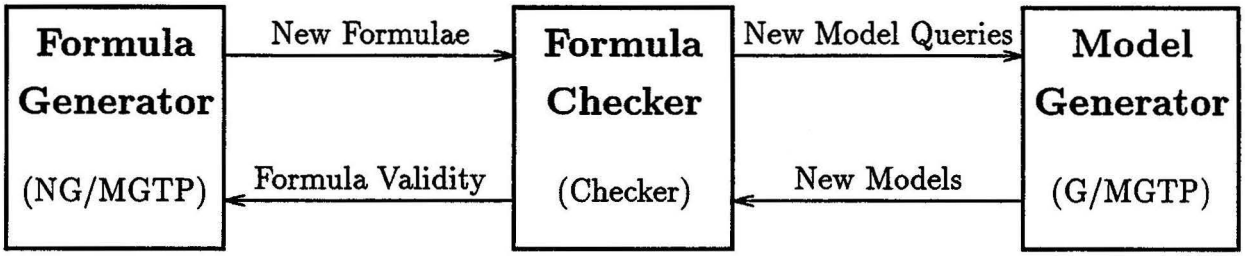


Figure 1: Semantic Constraint Configuration for MGTP

the axioms themselves admit many models. For resolution proving, the method of *semantic resolution* makes use of this.

Semantic resolution [1] is based on a well-known result that, given some model M, if there is a derivation of the empty clause from unification, resolution and factoring, then there is one in which no inference has parents that are both true in M. A given M divides all generated clauses into those that are true in M, and those that are not. A simple selection strategy can ensure that two clauses are not resolved together unless at least one is false in M. This is the strategy underlying semantic resolution, but may be applied equally well to model generation.

The value of such a strategy depends on the choice of M. An M for which many clauses are true is not as useful as an M for which fewer clauses are true. G/MGTP is well-suited to generating such models, for semantic use in N/MGTP.

A refinement on the process of semantic resolution is *semantic constraint*. A semantic constraint prover couples a model generator with the theorem prover, so that the models may change in response to exploration of the search space. Such a prover has been built as an adaptation to the resolution prover OTTER [7], and is now being developed for N/MGTP. The resulting program is called SC/MGTP.

3 Configuration

As shown in Figure 1, a semantic constraint prover has three components: The *Formula Generator* is a traditional theorem prover that generates inferences, but these are now filtered by a semantic constraint strategy. New inferences are passed to the checker for evaluation. The *Checker* evaluates the inference according to the current model, and determines whether a model update may be desirable. If so, the checker sends a set of constraints to the *Model Generator*, which responds with a new current model.

In order that completeness of this system not be sacrificed needlessly, the new models must validate all previously valid formulae, and so the constraints for new models increase monotonically, as do the models themselves. Thus, the cost of updating the model increases over time. A cut-off point may be set by the user to help control this cost.

4 Conclusions

At the time of writing, testing and algorithmic refinement of SC/MGTP are still underway. Tests conducted with SCOTT [8] show speed-ups of 2–3 times over sample problems. Similar improvements are expected for SC/MGTP.

The main significance in the thrust of this work is to produce a prover whose proof search is sensitive to its own computational history. In this approach, the prover continually refines its "understanding" of the problem, and uses this knowledge to direct further exploration of the proof space.

4.1 Future Work

Presently, the most significant limitation on the efficacy of this approach is in the cost of manipulating the models. In SC/MGTP the cost to check a formula in a model may be several times the cost to generate the formula in the first place, and the same problem occurs to a lesser extent in SCOTT. Model updates can be even more expensive. Investigations in how to store and work with models more efficiently are underway.

The MGTP provers are parallel provers, and N/MGTP in particular shows near–linear improvement as the number of processors increase. A future goal of this research is to streamline the checker to afford similar behaviour.

References

1. CHANG, C-L. AND LEE, R. C-T. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

2. FUJITA, H. AND HASEGAWA, R. *A Model Generation Theorem Prover in KL1 Using Ramified Stack Algorithm.* In *Proceedings of the Eighth International Conference on Logic Programming (Paris, France).* MIT Press, Cambridge, MA, 1991, 535–548.

3. HASEGAWA, R. *A Parallel Model Generation Theorem Prover: MGTP and Further Research Plan.* In *Proc. of the Joint American-Japanese Workshop on Theorem Proving*. Argonne, Illinois, 1991.

4. HASEGAWA, R. AND FUJITA, M. *Parallel Theorem Provers and their Applications.* In *Proceedings of the International Conference on Fifth Generation Computer Systems (Tokyo).* ICOT, Tokyo, 1992, 132–154.

5. HASEGAWA, R., KOSHIMURA, M. AND FUJITA, H. Lazy Model Generation for Improving the Efficiency of Forward Reasoning Theorem Provers. ICOT TR-751, 1992.

6. MANTHEY, R. AND BRY, F. *SATCHMO: A Theorem prover implemented in Prolog.* In *Proc. of CADE '88.* Argonne, Illinois, 1988.

7. MCCUNE, W. OTTER 2.0 Users Guide. Argone National Lab., ANL-90/9, Argonne, Ill., 1990.

8. SLANEY, J. SCOTT: A Model-Guided Theorem Prover. Submitted to: IJCAI-'93, October 1992.

A Tableau-Based Proof System for the Propositional μ-Calculus*

Michaela Huhn Peter Niebert
Universität Hildesheim
Institut für Informatik, Samelsonplatz 1
W-3200 Hildesheim, Germany
e-mail: {huhn|niebert}@informatik.uni-hildesheim.de

Abstract

The propositional μ-calculus L_μ is a decidable logic for the specification of properties of concurrent systems. The expressive power of L_μ results from the possibility to combine elementary modal and logical operators with solutions of fixpoint equations. We present a Gentzen-style proof system for the validity of implications of L_μ, which is based on syntactic fixpoint induction. It is derived from a tableau-based decision procedure for validity of a syntactically restricted class of L_μ given in [Koz 83]. In the system presented here, the class of admissible formulas is extended, but still restricted, and the fixpoint induction is directly incorporated into the proof system, not only in the soundness proof. While the latter modification yields larger worst-case bounds for the complexity of the resulting decision procedure, practical experiments indicate, that the algorithm is substantially faster on many examples. The proof system has been implemented in the *Concurrency Workbench*, a semantics tool for the analysis of CCS-processes.

1 Introduction

In a refinement approach for developing systems, starting from a loose specification, it is interesting to prove validity of implications $\Gamma \models \Delta$, where Γ denotes a more concrete, Δ a more abstract specification. $\Gamma \models \Delta$ then means that indeed Γ may be conceived as an instantiation of the more loose specification Δ; formally, the set of models of Γ is contained in the set of models of Δ.

The propositional μ-calculus L_μ is a modal logic introduced by Pratt [Pra 81] and Kozen [Koz 83], which can be used as a specification language for properties of concurrent programs. The modalities are used to speak about actions of parallel processes. Therefore L_μ is especially useful for action (rather than state) based models of concurrent systems such as CCS.

We present here a Gentzen-style proof system for sequents $\Gamma \vdash \Delta$, where Γ and Δ are sets of L_μ formulas. It is a extension of a tableau-based decision procedure for a restricted class of L_μ given in [Koz 83]. The main problem in such systems is the treatment of fixpoints. The proof rules dealing with fixpoints are based on an incremental fixpoint induction scheme. It works on the semantic level, i.e. the induction scheme speaks about sets of states in the models. It solves the problem to decide whether a given set of states

*The work presented here was carried out in the subproject C2 of the Sonderforschungsbereich 182 of the Univerity of Erlangen-Nürnberg

is included in a set satisfying a fixpoint formula without actually computing the exact solution of the fixpoint formula. The scheme works incrementally by expansions and modifications of the fixpoints by sets of states collected during a semantic proof. In the proof system working on the level of formulas rather than on the semantic level of state sets the semantic modifications of fixpoints are represented syntactically by attaching sets of modifying formulas (contexts) to fixpoints. Some technical problems arise which are solved by the incorporation of a priority method similar to one given in [Koz 83]. Using priorities avoids circular dependencies between fixpoints but the price to pay is a restriction of the set of admissible formulas which can be handled by our system.

The proof system is shown to be sound for L_μ, and to be complete for the restricted version of L_μ. The differences to the system given in [Koz 83] are discussed in section 7.

We have implemented the proof system in the environment of the *Concurrency Workbench* (CWB) for CCS and have experimental results on its usability.

2 Syntax and Semantics

The syntax of L_μ is given as follows: Let $Act = \{a, b, c, ...\}$ be a finite set of actions, $\mathcal{P} = \{P, Q, R, ...\}$ be a set of atomic propositions and $\mathcal{X} = \{X, Y, Z, ...\}$ be a set of propositional variables. We assume Act, $\mathcal{P}$, and $\mathcal{X}$ to be fixed for the rest of the paper. Then $\mathcal{F}$, the set of formulas of L_μ, is the smallest set such that

$$\mathcal{P}, \mathcal{X} \subseteq \mathcal{F}, \text{ and if } \varphi, \psi \in \mathcal{F}, \text{ then } \neg\varphi,\ \varphi \wedge \psi,\ [A]\varphi,\ \nu X.\varphi \in \mathcal{F}$$

where $A \subseteq Act$ and X may not appear negatively in φ in the formula $\nu X.\varphi$, i.e. every free occurrence of X is within an even number of negations. The free and bound variables of a formula are defined as usual, where ν binds variables. The operator '$[A]$' is called modality, 'ν' is called greatest fixpoint operator.

The following usual operators can be derived (and the other logical operators as well): $\langle A\rangle\varphi \equiv \neg[A]\neg\varphi$, and $\mu X.\varphi \equiv \neg\nu X.\neg(\varphi[X/\neg X])$, where $\varphi[X/\neg X]$ is the formula obtained by substituting all free occurrences of the variable X by its negation. 'μ' is called least fixpoint operator, and is responsible for the name of the L_μ for historical reasons.

L_μ is interpreted over transition systems $\mathcal{T} = (\mathcal{S}, trans, \mathcal{I})$, which consist of a set of *states* $\mathcal{S}$, a transition relation $trans \subseteq \mathcal{S} \times Act \times \mathcal{S}$, and an interpretation $\mathcal{I} : \mathcal{P} \to 2^{\mathcal{S}}$ of the atomic propositions as sets of states. For $(p, a, q) \in trans$ we also write $p \xrightarrow{a} q$.
A transition $p \xrightarrow{a} q$ means, that from state p we can reach state q by performing action a. The transition relation may be nondeterministic (for state p and action a there may exist distinct states q and q' with $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$). And the set of states may be infinite.
The atomic propositions can be interpreted as atomic properties of states, and we say that proposition P holds for state p iff $p \in \mathcal{I}(P)$. A valuation $v : \mathcal{X} \to 2^{\mathcal{S}}$ is an interpretation of the propositional variables as sets of states.

The semantics of a formula φ w.r.t. a transition system $\mathcal{T}$ and a valuation v is the set of states for which φ holds, and is denoted by $[\![\, \varphi \,]\!]_v^{\mathcal{T}}$ (where the indices $\mathcal{T}$ and v will be omitted when they are obvious). $[\![\, \varphi \,]\!]_v^{\mathcal{T}}$ is inductively defined by:

1. $[\![\, P \,]\!]_v^{\mathcal{T}} = \mathcal{I}(P)$, $[\![\, X \,]\!]_v^{\mathcal{T}} = v(X)$ for $P \in \mathcal{P}$, $X \in \mathcal{X}$
2. $[\![\, \neg\varphi \,]\!]_v^{\mathcal{T}} = \mathcal{S} \backslash [\![\, \varphi \,]\!]_v^{\mathcal{T}}$, $[\![\, \varphi \wedge \psi \,]\!]_v^{\mathcal{T}} = [\![\, \varphi \,]\!]_v^{\mathcal{T}} \cap [\![\, \psi \,]\!]_v^{\mathcal{T}}$

3. $[\![[A]\varphi]\!]^{\mathcal{T}}_v = \{p \in \mathcal{S} |\ \forall a \in A, q \in \mathcal{S} : p \xrightarrow{a} q \ \Rightarrow\ q \in [\![\varphi]\!]^{\mathcal{T}}_v\}$.

4. $[\![\nu X.\varphi]\!]^{\mathcal{T}}_v = \bigcup\{R \subseteq \mathcal{S};\ R \subseteq [\![\varphi]\!]^{\mathcal{T}}_{v[X/R]}\}$, where $v[X/R]$ is a valuation that equals v on all variables except for X and $v[X/R](X) = R$.

The defined set is the greatest fixpoint of the operator $\phi(R) = [\![\varphi]\!]^{\mathcal{T}}_{v[X/R]}$, i.e. the greatest set R, such that $\phi(R) = R$. We call ϕ **the operator induced by φ (in the variable X)**. The fixpoint property of $\bigcup\{R;\ R \subseteq \phi(R)\}$ is proven by the Tarski's fixpoint-theorem which requires that the operator ϕ is monotous in X. The monotony of the induced operator ϕ is guaranteed by the restriction that a variable X may only occur within an even number of negations in φ.
The semantics of the derived operators can be deduced from this, but for clarity we state $[\![\langle A\rangle\varphi]\!]^{\mathcal{T}}_v = \{p \in \mathcal{S} |\ \exists a \in A, q \in \mathcal{S} : p \xrightarrow{a} q \ \Rightarrow\ q \in [\![\varphi]\!]^{\mathcal{T}}_v\}$ and
$[\![\mu X.\varphi]\!] = \bigcap\{R \subseteq \mathcal{S};\ R \supseteq [\![\varphi]\!]^{\mathcal{T}}_{v[X/R]}\}$, i.e. $[\![\mu X.\varphi]\!]$ is the least fixpoint of the operator induced by φ.

3 An Example

In this section we discuss an example in detail which specifies behaviour of a channel modelled in terms of transition systems. The example is adapted from a temporal logic framework in [MaPn 92], where the discussed property is called *eventual reliability*. The motivation for the choice of the particular property is, that the used formulas do not satisfy the syntactic restriction in [Koz 83], but can be handled in the system presented here. It therefore illustrates the discussion of the restrictions in section 4.1. Also the example demonstrates the usage of fixpoint equations in specifications. For further examples we refer the reader to [Lar 90].

Our aim is to specify a channel, which receives a actions as input signals and sends b actions as output signals. The channel may be distorted, so that some inputs a can be ignored (lost), but that if the of input a is repeated sufficiently often, then there must be an output b. The behaviour of such a system is modelled by a transition system over the alphabet $\{a, b, \ldots\}$, and possible behaviours are described by paths through the system. Then the informal description of the channel is formalized by the requirement, that on paths with infinitely many a-transitions there must be at least one b-transition. In terms of transition systems we prefer to say *a is ultimately answered by b.*

The L_μ-specification is constructed in two steps, and for readability the properties are described in terms of paths. Formally an *A-path* ($A \subseteq Act$) in the transition system is a sequence of states $(p_i)_{i \in N}$ with transitions $p_i \xrightarrow{a_i} p_{i+1}$ and $a_i \in A$.

First we define the set of states p satisfying the following property: for all states q reachable from p by a finite number of A-transitions the atomic proposition P holds. The modality $[A]$ allows us to speak about single transitions, but speaking about paths needs fixpoint operators which allows us to define properties by equations of the form $[\![X]\!] = [\![\varphi]\!]$, where φ is positive in X. The equation $[\![Y]\!] = [\![P \wedge [A]Y]\!]$ describes the sets of states which satisfy: $p \in [\![Y]\!]$ iff p satisfies P and all $[A]$-successors of p are contained in Y. The equation holds for our property, but it does not uniquely specify $[\![Y]\!]$. However, any set of states $[\![Y]\!]$ solving the equation fulfils our property. Hence our property is the greatest fixpoint of the operator $\phi(R) = [\![P \wedge [A]Y]\!]^{\mathcal{T}}_{v[Y/R]}$. Thus $\nu Y.(P \wedge [A]Y)$ describes those states p of a transition system, for which P holds invariantly on all A-paths starting from p. The proposition P can be replaced by an arbitrary formula. So this property is closely related to the $\Box$-operator used in temporal logics.

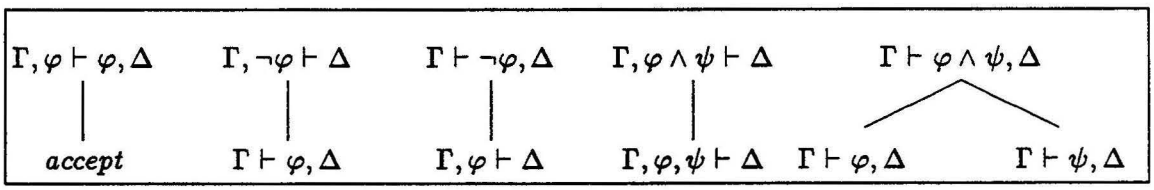
Now we want to specify the states p, such that on all paths starting from p and containing infinitely many a-transitions there is at least one b-transition. Dually we want to express that paths without b may only have finitely many occurences of a.
A state p fulfils this property iff all states q reachable on paths without b-transition and a single a-transition at the end also have this property, because a path from p infinitely many a-actions and without $b$actions would contain an infinite subsequence of such q's.
Using the formula defined above (by replacing A by $Act\backslash\{a,b\}$, and P by $[a]X$), we can translate this observation into the equation $[\![X]\!] = [\![\nu Y.[a]X \wedge [Act\backslash\{a,b\}]Y]\!]$. We omit to explain why the desired property is the least fixpoint of the equation, i.e. that it is contained in any other solution. The formula $\mu X.\nu Y.[a]X \wedge [Act\backslash\{a,b\}]Y$ specifies the states of a transition system for which infinitely many a-transitions are ultimately answered by a b-transition.

4 The Tableau Method

In this section we present our proof system for the validity of implications $\Gamma \models \Delta$. Such an implication is *valid* iff the states of arbitrary transition systems, which satisfy all of the formulas of Γ, satisfy at least one of formulas of Δ, i.e. iff for every transition system $\mathcal{T}$ and valuation v we have

$$\bigcap_{\gamma \in \Gamma} [\![\gamma]\!]_v^{\mathcal{T}} \subseteq \bigcup_{\delta \in \Delta} [\![\delta]\!]_v^{\mathcal{T}}.$$

We assume that only closed formulas occur in the implication $\Gamma \models \Delta$ and the names of the variables bound by the fixpoint operator ν are unique. So every variable X corresponds exactly to one fixpoint formula $\nu X.\xi$ and we denote the inner part of the fixpoint formula by ξ_X.
The proof system is a Gentzen-style sequent calculus: For proving the sequent $\Gamma \vdash \Delta$ a proof tree is build top-down according to a set of rules where the nodes are labelled with subgoals $\Gamma' \vdash \Delta'$. The rules either allow to append children with new subgoals to a node or to mark a node as an accepted (i.e. valid), as a rejected or as a not admissible leaf (i.e. a subgoal which cannot be handled). According to the syntax of the logic, three kinds of rules are distinguished: rules treating logical connectives, fixpoint rules, and rules treating the modalities. The success of the nodes of the tree is determined bottom up: leafs marked with *accept* are successful. Except for the rule for modalities, a goal is successful, iff its direct subgoals are successful. For the modal rule, things are a bit more complicated and will be discussed below.
The rules for treating the logical connectives and the "axioms" are standard:



The rules for fixpoints and modalities are discussed in the following sections.

4.1 Rules for Fixpoints

The treatment of fixpoint formulas on the right side of a sequent is based on an incremental version of Park's fixpoint induction principle [Par 70]:

Definition and Lemma: *Let S be a set, $R \subseteq S$ and $\phi : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$ a monotonous operator on the power set $\mathcal{P}(S)$ of S. Then the* **modified operator** ϕ_R *is defined by:*

$$\phi_R(S) := R \cup \phi(S) \qquad \text{for each } S \in \mathcal{P}(S)$$

Then:

1. *ϕ_R is monotonous and $\nu X.\phi(X) \subseteq \nu X.\phi_R(X)$.*
2. *$R \subseteq \phi(\nu X.\phi_{R \cup S}(X))$ implies $R \subseteq \nu X.\phi_S(X)$.*
3. *If $R \subseteq S$ then $R \subseteq \nu X.\phi_S(X)$.*

Similar versions of this lemma with proofs can be found in [Win 89] and [Cle 90].

This lemma allows us to apply an incremental fixpoint induction scheme for semantic proofs: If we want to prove that a set R of states satisfies a formula $\nu X.\xi_X$, i.e. if $R \subseteq \nu X.\phi$ where $\phi(M) := [\![\xi_X]\!]^{\mathcal{T}}_{v[X/M]}$, then by (2) we can reduce this to a proof for: $R \subseteq \phi(\nu X.\phi_R)$. This may eventually (by decomposition of the operator ϕ) lead to a goal $S \subseteq (\nu X.\phi_R)$. Then again by (2) we can add the subset S to the modified operator and we get $S \subseteq \phi(\nu X.\phi_{R \cup S})$. After adding some more modifications to the fixpoint operator we may reach a subgoal $Q \subseteq \nu X.\phi_T$ with $Q \subseteq T$. Then by (3) this assertion is true.
By this scheme the meaning of fixpoint is reduced to the meaning of the operator and the modifications of the operator. The process of replacing $\nu X.\phi_R$ by $\phi(\nu X.\phi_{R \cup S})$ is also called unrolling of the fixpoint.

This semantic fixpoint induction scheme can be transformed to a proof scheme working on the syntactic level of sequents. Let us consider a sequent (subgoal) $\Gamma \vdash \Delta, \nu X.\xi_X$ where a fixpoint formula $\nu X.\xi_X$ occurs on the right hand side of the sequent. The sequent is valid iff

$$]\!](\Gamma;\Delta)[\!]^{\mathcal{T}}_v := \bigcap_{\gamma \in \Gamma} [\![\gamma]\!]^{\mathcal{T}}_v \cap (\mathcal{S} \setminus \bigcup_{\delta \in \Delta} [\![\delta]\!]^{\mathcal{T}}_v) \subseteq [\![\nu X.\xi_X]\!]^{\mathcal{T}}_v,$$

where $\mathcal{S}$ is the set of states in the model $\mathcal{T}$. We call $(\Gamma;\Delta)$ the **context** of the of the fixpoint formula, and $]\!](\Gamma;\Delta)[\!]^{\mathcal{T}}_v$ can be understood as the set of witnesses against the validity of the sequent without the fixpoint. From the fixpoint lemma results the following proof rule: We unroll the fixpoint formula $\nu X.\xi_X$ and obtain the subgoal $\Gamma \vdash \Delta, \xi_X(X/\nu X_{(\Gamma;\Delta)}.\xi_X)$.

The indexed variable $X_{(\Gamma;\Delta)}$ syntactically represents that the fixpoint operator corresponding to the subformula ξ_X is modified by the set $]\!](\Gamma;\Delta)[\!]$. For iterated unrollings lists of contexts decorating the fixpoint variables are the syntactic equivalent to the union of modification sets indexing the semantical operators.
After applying several more proof rules we may reach a sequent $\Gamma \vdash \Delta, \nu X_{Con_X}.\xi_X$ where the actual context $(\Gamma;\Delta)$ equals (syntactically) one of the contexts in the context list Con_X. Then the fixpoint induction was successful.

However this syntactic representation of modifying sets raises some problems: the contexts themselves might contain modified fixpoints and so on. This could lead to an unbounded number of different formulas in a tableau, and therefore possibly infinite tableaux. Moreover if formulas which differ only in the modifications are distinguished in sequents, then sequents may become arbitrarily large, and this is another source of possibly infinite tableaux. To avoid this the aim is to bound the nesting depth of modifying lists. This can be achieved by forcing a total order on the modified fixpoints in the sequent.

For this we adopted the priority method used in [Koz 83]: *Each time an unmodified fixpoint is modified, we attach the least natural number which is not already used as a*

priority in the sequent to the fixpoint variable as its actual priority (a low number means high priority). The invariant for the lifetime of a context list is, that in the contexts only fixpoints with higher priorities are modified. In addition, there may not appear two formulas on the same side of a sequent, which differ only in the modifications.

We attach the priority assigned to a fixpoint variable as an index: So the sequent $\Gamma \vdash \Delta$ consists of formulas φ where the subformulas $\nu X.\xi_X$ may be extended by a priority and a context list, namely we have $\nu X_{(n,Con_X)}.\xi_X$.

Example: Let us consider the formula $\varphi = \nu X.\nu Y.X \wedge Y$ which may occur within a sequent $\Gamma \vdash \varphi, \Delta$). When the outermost fixpoint is unrolled for the first time we get as new subgoal the sequent

$$\Gamma \vdash \nu Y.(\nu X_{(n,(\Gamma;\Delta))}.(\nu Y.X \wedge Y)) \wedge Y, \Delta$$

where n is the least natural number not used in the context. Next we may unroll the inner fixpoint and obtain

$$\Gamma \vdash \nu X_{(n,(\Gamma;\Delta))}.(\nu Y.X \wedge Y) \wedge \nu Y_{(n+1),(\Gamma;\Delta))}.(\nu X_{(n,(\Gamma;\Delta))}.(\nu Y.X \wedge Y) \wedge Y), \Delta.$$

Now the formula may be decomposed according to the $\wedge$-rule and in the left subgoal

$$\Gamma \vdash \nu X_{(n,(\Gamma;\Delta))}.(\nu Y.X \wedge Y), \Delta$$

the priority $n+1$ has disappeared and can be used again. However this goal can already be recognized as valid, since the context $(\Gamma;\Delta)$ appears in the context list of X.

The invariant that contexts modifying fixpoints with priority n may only depend on modifications of fixpoints with higher priorities has some consequences for the proof rules: when a modified fixpoint is unrolled in a context, where also fixpoints with lower priorities are modified, then their modifying context lists must be emptied. This does not affect the correctness of the rule, since modifications only increase the set of states that satisfy a sequent. Second, if by some decomposing proof rule two formulas, which differ only in the modifications, appear in a sequent, then the one with the highest priority not appearing in the other is taken, and the other is discarded. The discarding of formulas does not affect the correctness of the system. Finally, if somehow a fixpoint with a certain priority n vanishes from a sequent, then the modifications of lower priority can be discarded as well, since they are certain to depend syntactically on the fixpoint with priority n, i.e. they would never lead to a successful fixpoint induction. Then the lower priorities are raised to fill out the empty places. Thus by and by the priority of a certain fixpoint rises, so only a finite number of other fixpoints can be responsible for the clearing of its context list. This is needed for the completeness proof, where fixpoints are finitely approximated.

However here we meet a **restriction**: by the decomposition of formulas with the left $\wedge$-rule it is possible, that the same priority of a fixpoint to be unrolled appears in another formula. In this case we would obtain two fixpoints of the same priority, but with different modifications, and the ordering by the priority would be lost for these two fixpoints, leading to troubles with finiteness or completeness. In [Koz 83] this problem is avoided by restricting the set of formulas in such a way, that a fixpoint with the same priority may not appear in both parts of a formula $\varphi \wedge \psi$ on the left side of a sequent. The corresponding syntactic restriction is called *aconjunctivity*. While this is a sufficient restriction to avoid the situation, it is not neccessary, since the modal rule to be described below may split formulas into different sequents, when they are headed by different modalities.
Therefore we just break off at such a forbidden situation without a result. A syntactic

characterization taking the splitting of formulas by the modal rule into account is possible, but complicated.

Concluding, we obtain the following proof rules:

$$\frac{\Gamma \vdash \nu X.\xi_X, \Delta}{\Gamma \vdash \xi_X(X/\nu X_{(n,\{(\Gamma;\Delta)\})}.\xi_X), \Delta}$$

if X occurs without a priority and $m \in \mathbf{N}$ is the lowest priority not already used in the sequent.

$$\frac{\Gamma \vdash \nu X_{(n,Con_X)}.\xi_X, \Delta}{\Gamma' \vdash \xi_X(X_{(n,Con_X)}/\nu X_{(n,Con_X \cup \{(\Gamma';\Delta')\})}.\xi_X), \Delta'}$$

if the context $(\Gamma';\Delta')$ is not contained in Con_X, where Γ' and Δ' denote the formulas in the sequent where the modifications of priority lower than n have been erased.

$$\frac{\Gamma \vdash \nu X_{(n,Con_X \cup \{(\Gamma';\Delta')\})}.\xi_X), \Delta}{accept}$$

if the context $(\Gamma;\Delta)$ is contained in Con_X.

Dealing with fixpoints on the left hand side

The proof rules for left hand side fixpoints are completely different:
In a sequent $\Gamma, \nu X.\xi_X \vdash \Delta$ the fixpoint $\nu X.\xi_X$ is just unrolled: we get the subgoal $\Gamma, \xi_X(X/\nu X.\xi_X) \vdash \Delta$. The fixpoint property guarantees, that this does not affect the semantics. Now we must take care that fixpoints cannot be unrolled infinitely often. So we only permit to unroll the fixpoint $\nu X.\xi_X$ a second time if its semantics has been changed by the fixpoint rules for the right side or after an application of the modal rule. This condition is implemented by a flag which is reset if a prior fixpoint is modified. The idea of this technical condition is that multiple unrollings of a fixpoint will not reveal more information useful for proofs in other situations. Showing this is part of the completeness proof.

$$\frac{\Gamma, \nu X.\xi_X \vdash \Delta}{\Gamma, \xi_X(X/\nu X_m.\xi_X) \vdash \Delta}$$

if X has not yet been unrolled since the last application of the modal rule, or if the semantics has changed since the last unrolling.

4.2 The Modal Rule

The modal rule, i.e. the rule for treating formulas of the form $[A]\varphi$, is only allowed if no other rule (for logical connectives or fixpoints) is applicable, has the following form:

$$\frac{\Gamma \vdash \Delta,\ [A_1]\varphi_1,\quad [A_2]\varphi_2,\quad \ldots\quad [A_m]\varphi_m}{\Gamma_1^1 \vdash \varphi_1\ \ \cdots\ \ \Gamma_{n_1}^1 \vdash \varphi_1 \quad \Gamma_1^2 \vdash \varphi_2\ \ \Gamma_{n_2}^2 \vdash \varphi_2 \quad \Gamma_1^m \vdash \varphi_m\ \ \cdots\ \ \Gamma_{n_m}^m \vdash \varphi_m}$$

where $A_i = \{a_1^i, \ldots, a_{n_i}^i\}$, Δ contains no modal formulas, and $\Gamma_j^i = \{\psi;\ \exists B \subseteq \mathcal{A} : [B]\psi \in \Gamma$ and $a_j^i \in B\}$; the modal rule is only applicable if the same sequent $\Gamma \vdash \Delta, [A_1]\varphi_1, \ldots [A_m]\varphi_m$ has not already occurred on the path between the root of the tableau and the actual subgoal otherwise we add *reject* as a child. This is a breakoff rule that ensures the finiteness of the tableaux.

Note that for each modal formula on the right hand side we get a bundle of children, one child for each action $a_j^i \in A_i$. The labels a_j^i on the edges of the tableau are used for the construction of a witness of the invalidity of $\Gamma \vdash \Delta$ in the case of an unsuccessful tableau. The use of the generalized modality $[\{a_1, \ldots, a_k\}]\varphi$ instead of $[a_1]\varphi \wedge \ldots \wedge [a_k]\varphi$ allows the more efficient rule for treating modalities in Gentzen style systems, which takes advantage of the special form of this conjunction: replacing the generalized modalities by combinations of simple modalities and $\wedge$ in the parent of the given rule would result in $n_1 \cdot \ldots \cdot n_m$ children instead of $n_1 + \ldots + n_m$. If none of these rules is applicable we give up and add the leaf *reject*.

As stated the success of a goal at the application of the modal rule is defined in a different way than for the other rules: The parent node of the application of the modal rule is succsessful iff at least one of the bundles contains only successful children. To understand, why this rule is correct, let us consider the case that all children of the first bundle are valid. Let p is a state satisfying all of Γ, and $R_i = \{q; p \xrightarrow{a_i^1} q\}$ be the set of a_i^1-successors of p. By the interpretation of the modality operator all states of this set satisfy all formulas of Γ_i^1. By the validity of the corresponding subgoal $R \subseteq [\![\varphi_1]\!]$, i.e. all a_i^1-successors of p satisfy φ_1. Since this holds for all $a_i^1 \in A_1$, we get $p \in [\![[A_1]\varphi_1]\!]$, thus showing the validity of the sequent.

5 Soundness, Finiteness, and Completeness

The proof system is sound in the sense, that a sequent $\Gamma \vdash \Delta$ is valid if there exists a successful tableau for it. It is complete for the restricted subset of L_μ in the sense that the sequent is not valid, if there exists an unsuccessful tableau for it. Additionally it is guaranteed, that every tableau generated by the proof rules is finite. This yields a nondeterministic decision procedure: successively apply the proof rules to a tableau starting with the initial sequent as the root, until all leafs are marked with *accept* or *reject*, then determine the success of the root.

The soundness of the proof system follows from the backward soundness of the proof rules, and the validity of successful leafs. Arguments for the backward soundness of the important proof rules were given in the last section.

We only give a skeleton of the proofs neccessary to show that the generated tableaux are finite and the proof system is sound, i.e. that the root of each succesful tableau is valid. It is easily seen that the set of formulas $\varphi \in L_\mu$ which can be generated in a tableau by these proof rules is finite, because all occuring formulas are contained in the Fischer-Ladner-Closure [FiLa 79] of the initial sequent $\Gamma \vdash \Delta$. Moreover the lowest of priority which may be distributed to a fixpoint in the tableaux is limited by $|\Gamma \cup \Delta|^2$ where $|.|$ denotes the number of symbols in a set of formulas.
From the finite number of priorities and occuring subgoals $\Gamma \vdash \Delta$ we conclude that for each fixpoint variable the cardinality of the context list is bounded. We need some additional techniques to show that the proof rules concerning the erasion of context lists due to the modification of higher priority variables do not introduce infinite unrolling of fixpoints. The main idea is to observe the behaviour of a fixpoint variable along a path and count the

number of all modifications in the priority and context list of the selected variable. Starting with the variable of priority 1 we can proof inductively that the number of modifiactions must be finite between two applications of the modal rule for all variables. Because there are only finitely many sequents and a repitition of the same sequent leads to a break off, the number of applications of the modal rule is limited as well. Thus all tableaux generated by the proof rules are finite.

The completeness proof for the proof system largely resembles the one given in [Koz 83], therefore we just sketch the main ideas and the differences. Completeness is shown by explicitly constructing a transition system from an unsuccessful tableau, which is a witness against the validity of the sequent.

The construction is done in several steps: first the tableau is reduced to a minimal unsuccessful skeleton, i.e. for nodes with several subgoals a minimal set of (unsuccessful) subgoals is chosen, which ensures, that the goal itself is unsuccessful. For the right $\wedge$-rule, this is one of the two successors, for the modal rule this is one successor of each bundle. This process is performed topdown, until we obtain a tree, which branches only at places, where the modal rule was applied. From this skeleton a transition system is build, taking as states those nodes, where the modal rule was applied. Transitions are defined as follows: If a path leads from a modal node n to a modal node m with no modal nodes on the way, and the first edge is labelled by action a, then an a-transition leads from the state corresponding to n to the state corresponding to m. Differing from the procedure given in [Koz 83], which treats infinite tableaux, we have a breakoff rule which ensures that the resulting tableaux, and therefore also the constructed transition systems are finite. Thus special care has to be taken for the breakoffs. The breakoff rule is applied at a node m, when a modal node m' appears above (on the way to the root) in the tableau with an identical set of formulas and context lists. In this case a path from a modal node n to m as described above leads to a transition from the state corresponding to n to the state corresponding to m' instead of m. In other words, the breakoff rule leads to backward loops.

After the construction of the transition system it is shown, that a state corresponding to a modal node n (roughly) fufills all formulas on the left side and none on the right side of sequents on the ways from its preceding states (modal nodes). This means, that the state is a witness against the validity of these sequents. In particular we find such a state for the initial sequent.

6 Implementation and Practical Results

We have implemented the proof system in the framework of the *Concurrency Workbench* [CPS 89], a package that integrates a variety of tools for the analysis of CCS-processes and includes a model checker for L_μ.

We have experimented with the prototype and proved a number of examples including some known implications which were translated from other logics such as linear and branching time temporal logic. For illustration we give an example below. In most cases the proofs were very small even for properties requiring several nested fixpoints, but not surprisingly also we found small examples (with only two nested fixpoints) which resulted in very large proofs.

We have used some heuristics to improve the algorithm and evaluated these by modifying the prototype. These include an efficient representation of sets of formulas, and strategies for the application order of the rules, that lead to smaller tableaux. In particular, the

information in the context lists as given above is very redundant, so that we used equivalent representations which are much smaller and allow efficient comparisons.

Since a tableau generally contains many identical subtableaux the most rewarding improvement of the algorithm is the maintenance of a database with intermediate results. On the theoretical side, this technique relates to the simulation technique for alternating turing machines proposed in [CKS 81], which is also used in [Koz 83]. This approach results in turning a bound for the *depth* of the proof trees into a bound for the *size* of reduced proof trees. In our experiments, the increase in speed resulting from this technique was impressing. Since the amount of data to be stored can be huge, efficiency of the implementation in both time and space requirements are crucial. Indeed, space may become the critical resource, and some restriction strategy concerning the results to be stored should be considered.

As an example we investigate proofs concerning the formulas of section 3. We assume $Act = \{a, b, c\}$. For easy reading we introduce some shorthand notations or macros for formulas: $UA(x, y) := \mu X.\nu Y.[x]X \wedge [Act \backslash \{x, y\}]$, where $x, y \in Act$; as discussed in section 3, this property holds for some state p, if any path starting from p with infinitely many x has at least one y. $\Box\varphi := \nu X.\varphi \wedge [Act]X$, where φ is an arbitrary formula with no free occurrence of the variable X; this holds for some state p, iff φ holds for all states reachable from p.

Now we investigate some implications between formulas constructed from these macros, which are variations over the question: $(\Box)UA(a, b)$, $(\Box)UA(b, c) \models (\Box)UA(a, c)$? The idea is to understand, under what conditions the relation "x is ultimately answered by y" is transitive. To understand the results intuitively we note, that $\Box\, UA(x,y)$ means, that on a path with infinitely many occurrences of x there must also be *infinitely* many occurences of y (in constrast to just one occurrence of y).

For each investigated sequent we give the result of the proof attempt, the total tableau size (TS), the execution time (ET) on a SparcStation 10/20, the number of symbols in the initial sequent (SS), the number of introduced context lists (LI), the number of times any context list has been appended (LA), and the number of successful fixpoint inductions (FI).

sequent	valid	TS	ET	SS	LI	LA	FI
$UA(a,b), UA(b,c) \vdash UA(a,c)$	no	55	0.03s	21	5	8	1
$UA(a,b), \Box UA(b,c) \vdash UA(a,c)$	no	97	0.04s	25	9	12	1
$\Box UA(a,b), UA(b,c) \vdash UA(a,c)$	yes	194	0.01s	25	14	25	6
$\Box UA(a,b), UA(b,c) \vdash \Box UA(a,c)$	no	728	0.17s	29	52	95	18
$\Box UA(a,b), \Box UA(b,c) \vdash \Box UA(a,c)$	yes	3322	1.13s	33	279	405	54

The last three columns show, that the context lists were always very small in this example. For instance for the last row it follows, that successful fixpoint inductions occurred with lists of no more than 9 entries on average, and 405 is a secure upper bound for the length of the lists.

While this observation, that the context lists are small, was done on most examples we tested, it should be mentioned, that we also found sequents of comparable size, which require much larger proofs. The reason for this however, never seemed to result from the fixpoint induction, but from a certain form of nondeterminism in the formulas, which lead to large case analyses.

7 Discussion and Related Work

Our proof system is based on a tableau method given in [Koz 83]. Our system differs from that system in several (independent) ways:
The subset of L_μ for which our proof system is shown complete (let us call it the set of admissible formulas) is an extension of the set of *aconjunctive formulas* given as a restriction of L_μ for the system in [Koz 83]. For instance the second example in section 3 is not aconjunctive, but can be handled in our system. Aconjunctivity is a syntactic restriction of the set of formulas, which is not closed under negation, which makes it difficult to analyse the expressive power of this subset of L_μ. If we require closure under negation, i.e. if we regard the subset of aconjunctive formulas φ for which $\neg\varphi$ is also aconjunctive, then it is possible to show that the set of formulas φ such that both φ and $\neg\varphi$ are admissible, is strictly more expressive. The reason for this is, that the resulting restriction of aconjunctive formulas actually excludes the nesting of fixpoints, i.e. in a formula $\nu X.\varphi$ there may be only one free occurrence of the variable X in φ, and this must not be within a fixpoint subformula.

Furthermore, in contrast to [Koz 83], the proof rules generate only finite tableaux, and in case of an unsuccessful tableau, a finite model that proves the invalidity of $\Gamma \vdash \Delta$ is explicitly constructed.

The system in [Koz 83] uses counters instead of lists of syntactic contexts for handling the fixpoints. The number of fixpoint expansions is counted until a repetition of the context must have occurred, which is ensured by making a safe estimation of the number of contexts a formula may appear in. The estimation used in [Koz 83] is 2^n, where n is the number of symbols in the initial sequent. This estimation assumes, that the representation of the modifications could be flattened in such a way, that the contexts do not contain modifications, and that these modifications could be inferred by the priorities and the modifications of the other fixpoints. However the formulas in the sequents must a least contain priorities (otherwise this representation is ambiguous), but this does not add substantially to the complexity of the algorithm: this methods yields an (in the number of symbols of the sequent) exponential algorithm for the worst case, and also for the best case as soon as fixpoints are involved.
The theoretical bound we were able to show by the same methods is double exponential for the worst case. But our experiments indicate that the fixpoint induction acts in a "local" way, i.e. fixpoints are mostly expanded in similar contexts. For instance in the last example of the previous section, with a sequent of 33 symbols, the length of a context lists was at most 9 on average and certainly no more than 405 in any case (probably much less), while the counter method would require at least 2^{33} steps for each successful fixpoint induction.
Hence the context lists are very short and the same context occurs soon compared to the theoretical bounds, or conversely the estimation used in the counter method is too generous. This implies that in many cases the tableaux we obtain are substantially smaller than those generated by the counter method.

The method to describe the semantics of a fixpoint expansion by defining a semantic modification of the operator corresponding to a syntactic construct is used in a different framework in [Cle 90] and [Win 89].
A double exponential decision procedure for full L_μ, which is based on automata theoretic constructions rather than logical reasoning, is given in [StEm 89].

Acknowlegdements

We thank Norbert Götz for his support of our work in many ways. As for this paper we thank the anonymous referees for their comments and suggestions, and Heike Wehrheim for discussions of the examples.

References

[CKS 81] A. Chandra, Dexter Kozen and Larry Stockmeyer, Alternation, in *Journal of the Association of Computing Machinery* 28(1) (1981), 114-133.

[Cle 90] Rance Cleaveland, Tableau-Based Model Checking in the Propositional Mu-Calculus, in *Acta Informatica* 27 (1990), 725–747.

[CPS 89] Rance Cleaveland, Joachim Parrow and Bernhard Steffen, *The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems*, Report ECS-LFCS-89-83, University of Edinburgh, August 1989.

[FiLa 79] Michael J. Fischer and Richard E. Ladner, Propositional dynamic logic of regular programs, in *Journal of Computer and System Sciences* 18, 1979, 194–211

[HuNi 92] Michaela Huhn and Peter Niebert, *Realisierung eines Beweissystems fuer Hennessy-Milner Logik mit Rekursion*, Universität Erlangen 1992, Diplomarbeit

[Koz 83] Dexter Kozen, Results on the propositional μ-calculus, in *Theoretical Computer Science* 27 (North Holland 1983), 333–354.

[Lar 90] Kim G. Larsen, Proof systems for satisfiability in Hennessy-Milner Logic with recursion, in *Theoretical Computer Science* 72 (North Holland 1990), 265–288

[MaPn 92] Zohar Manna and Amir Pnueli, *The temporal logic of reactive and concurrent systems*, Springer 1991

[Par 70] D. M. R. Park, Fixpoint induction and proof of program semantics, in B. Meltzer and D. Michie, eds., *Mach. Int. 5* (Edinburgh Univ. Press, 1970) S. 59 - 78

[Pra 81] V. R. Pratt, A decidable mu-calculus: preliminary report, in *Twenty-Second Ann. Symp. on Foundations of Computer Science*, Nashville, Tennessee 1981, 421–427

[StEm 89] Robert S. Streett and E. Allan Emerson, An Automata Theoretic Decision Procedure for the Propositional Mu-Calculus, in *Information and Computation* 83(3) (1989), 249–264.

[Win 89] Glenn Winskel, Model Checking in the Modal ν-Calculus, in *Proceedings of Eleventh International Colloquium on Automata, Languages and Programming* 1989

Circumscription and Minimal Models for Propositional Logics

Pascale Kuhna

G.I.A., Faculté des Sciences de Luminy,
163, Avenue de Luminy, Case 901,
13288 Marseille, Cedex 9, France
phone: (33) 91 26 93 14
e-mail: kuhna@gia.univ-mrs.fr

22 Mars 1993

Abstract

Minimal entailment is the semantical counterpart of Circumscription and Closed World assumption. In this paper, we are interested in circumscription where a predicate is allowed to vary and we show how to find the whole set of minimal models for a propositional theory, using semantical tableaux method. This research allows us to explicit the connection between the semantical and the syntactical approach of circumscription and we will see, with an example, how to deduce interesting formulae, which can be deduced in a syntactical way, using minimal models.

1 Introduction

McCarthy's theory of circumscription appears to be the most powerful among various formalisations of nonmonotonic reasoning designed to handle incomplete and negative information in knowledge representation systems. Moreover, one of its major advantage over the other approaches is the fact that circumscription is based on classical predicate logic: the initial theory is completed with a diagram of axioms called circumscription diagram. This globally expresses that the conditions which make true the circumscribed property P are necessary for validating P. Obviously, these conditions have to be compatible with the rules of the theory. Nevertheless, it is not so easy to apply the circumscription diagram because the conditions when the property P appears are found in an intuitive way: we will illustrate this problem in section 1 with an example from [LEA SOMBE,1990]. In this paper, we describe the semantical approach of circumscription. It is based on a predicate minimization which consists in considering for a theory only models where less circumscribed predicates appear. The notion of minimal entailement is naturally introduced and a formula f is minimally entailed from a theory if it is true in all the minimal models of the theory. [OLIVETTI,1992] uses the analytic tableaux method to formalize this minimal entailement but he only evokes the minimal entailement with fixed atoms. By studying this particular minimization, we will add to his research. Moreover, our approach is quite different from his: the question is not to know if a formula f is minimally entailed from a

theory, but to search for the set of minimal models of the theory. This reseach will show us how the minimal conditions when the predicate P appears are induced and restrincting ourselves to these minimal conditions, we will produce some interesting deductions. We have, here, an approach close to the syntactical approach we have described previously. We will illustate it with the example of [LEA SOMBE,1990] in section 4. Before this, in section 2, the analytic tableaux method is presented. It is well adapted for seaching minimal models because it gives, for a theory, representative classes of models which will be completed in a particular way in order to obtain minimal models. This is in section 3.

2 Circumscription

2.1 Syntactical definition

Let $\mathcal{B}$ be a first order theory that contains both the predicate symbols P and Q. The circumscription consists of a diagram of axiom called, a circumscription diagram, which extends the proof theory for first order logic. The circumscription diagram of the predicate symbol P with Q allowed to vary is:

$$(\mathcal{B}[\phi_P, \varphi_Q] \wedge (\forall x\, \phi_P(x) \rightarrow P(x))) \rightarrow (\forall x\, P(x) \rightarrow \phi_P(x))$$

- ϕ_P and φ_Q can be any formula.
- $\mathcal{B}[\phi_P, \varphi_Q]$ is the set $\mathcal{B}$ where each occurrence of P and Q is substituted respectively for ϕ_P and φ_Q.
- The set of all instances of this diagram is written: $Circums_B[P, Q]$.

A formula f derives from circumscription if $\mathcal{B} \cup Circums_B[P, Q] \vdash f$.

We note that all the instances of the circumscription diagram belong to the theory and can be used. But most of them will be inefficient. The real problem is in choosing the "right" intances for ϕ_P and φ_Q: we must have an accurate idea about what can be deduced and how to deduce it. We will see that with the semantical approach of the circumscription, using minimal models, we will not only be able to decide if a formula derives from the circumscription but we will also characterize where circumscribed predicates appear and thus, have an idea of the kind of formulae that could be proved. The next example from [LEA SOMBE,1990] illustrates the syntactical deduction of circumscription. It will be dealt in section 4 and we will see how the minimal model research will help us to find the right instances for ϕ_P and φ_Q.

2.2 Example [LEA SOMBE,1990]

The example is expressed as follows: every student that is not abnormal is young. Lea is a student and Paul is not young. This can be translated into first order logic:

$$\forall x\, Student(x) \wedge \neg abnormal(x) \rightarrow Young(x)$$

$$Student(Lea) \wedge \neg Young(Paul)$$

The predicate *abnormal* is circumscribed and the predicate *Young* varies. To circumscribe the predicate *abnormal* means that we will determine the conditions where *abnormal* appears and we will deduce that it only appears in these conditions. In the example, this allows us to conclude that abnormality only refers to *Paul* when he is considered as a student. Thus, we will deduce that the other students are young. Such a deduction would be impossible in first order logic.

Let us apply the circumscription diagram $Circums_B[abnormal, Young]$ to the theory:

$$([Student(Lea) \wedge \neg\varphi_{Young}(Paul) \wedge (\forall x\ Student(x) \wedge \neg\phi_{abnormal}(x) \rightarrow \varphi_{Young}(x))] \wedge [\forall x\ \phi_{abnormal}(x) \rightarrow abnormal(x)]) \rightarrow [\forall x\ abnormal(x) \rightarrow \phi_{abnormal}(x)]$$

We chose the instance: $\phi_{abnormal}(x) : Student(x) \wedge x = Paul$
$\varphi_{Young}(x) : x \neq Paul$

and we replace $\phi_{abnormal}(x)$ and $\varphi_{Young}(x)$ in the previous formula.

The premises of the instance derive from the theory

thus: $\forall x\ abnormal(x) \rightarrow (Student(x) \wedge x = Paul)$
moreover: $\forall x\ Student(x) \wedge \neg abnormal(x) \rightarrow Young(x)$

We conclude that:

$$\forall x\ Student(x) \wedge x \neq Paul \rightarrow Young(x)$$

2.3 Semantic

The syntactical definition is associated with a natural semantical definition: we will only consider models of the theory where less individuals satisfy the cicumscribed predicates, while respecting rules of the theory. We introduce a preorder relation in order to compare models and the smallest ones, that [BESNARD,SIEGEL,1989] call preferential models, will be used to make deductions. Let us study Lea's situation in the above example. We only know that she is a student and with no further information about her, we conclude that she is not abnormal and thus, that she is young. The model that contains $\neg abnormal(Lea), Student(Lea)$ and also $Young(Lea)$ is preferred to a model that contains $abnormal(Lea)$ and $Student(Lea)$. We introduce a preorder relation on the models of the theory in order to reduce the cases of abnormality to a minimum.

Definition 1 *Let $\mathcal{B}$ be a first order theory that contains the predicates P and Q, P being circumscribed and Q being allowed to vary. Let m and m' be two models of the theory $\mathcal{B}$. $|P|_m$ denoting the extension in m of P, we define the relation $\leq_{P;Q}$ such that:*

$$m \leq_{P;Q} m' \text{ iff } \begin{array}{l} |P|_m \subseteq |P|_{m'} \\ m \text{ and } m' \text{ are absolutely identical anywhere else} \\ \text{except on } |Q|_m \text{ and } |Q|_{m'} \end{array}$$

This relation is reflexive and transitive but not antisymetric. For example, if a model m of $\mathcal{B}$ coincides with a model m' of $\mathcal{B}$ except on $|Q|_m$ and $|Q|_{m'}$, we have both $m \leq_{P;Q} m'$ and $m' \leq_{P;Q} m$. Thus, we have to alter the usual definition of minimality for an order relation: a model m of $\mathcal{B}$ will be minimal for $\leq_{P;Q}$ if there are no models of $\mathcal{B}$ "absolutely" smaller than m.

Definition 2 *A model m of $\mathcal{B}$ is minimal for $\leq_{P;Q}$ iff for each model m' if $m' \leq_{P;Q} m$, then $m \leq_{P;Q} m'$.*

3 Models

3.1 Analytic tableaux method

The analytic tableaux method was first devised to prove the validity of theorems. It is used here to find the set of models of the theory $\mathcal{B}$. [SCHWIND,1990] describes the complete method using the operator TP. TP associates a set of formulae with sets of literals and is defined recursively such that:

$TP(M) = TP(M' \cup \{l\})$	if $\neg\neg l \in M$ and $M' = M \backslash \{\neg\neg l\}$
$TP(M) = TP(M' \cup \{l\} \cup \{l'\})$	if $l \wedge l' \in M$ and $M' = M \backslash \{l \wedge l'\}$
$TP(M) = TP(M' \cup \{\neg l\}) \cup TP(M' \cup \{\neg l'\})$	if $\neg(l \wedge l') \in M$ and $M' = M \backslash \{\neg(l \wedge l')\}$
$TP(M) = TP(M' \cup \{A(a)\})$	if $\exists x\, A(x) \in M$ where a is a new parameter and $M' = M \backslash \{\exists x\, A(x)\}$
$TP(M) = TP(M' \cup \{A(a)\})$	if $\forall x\, A(x) \in M$ where a is any parameter and $M' = M \backslash \{\forall x\, A(x)\}$
$TP(M) = \{M\}$	if M is a set of literals.

TP associates a formula, by decomposing it with the previous rules, with its disjunctive normal form: each set represents the conjunction of its elements, the tableau represents the disjunction of its elements. For example:

$$TP(\{(a \wedge b) \vee c\}) = \{\{a, c\}, \{b, c\}\} = TP(\{(a \wedge c) \vee (b \wedge c)\})$$

The tableau TP may contain closed sets or sets of literals such that a part of the set is a set of TP itself. Thus, we consider a minimal form TS of TP such that:

$$TS(M) = TP(M) \backslash \{\ X : X \in TP(M) \text{ and } X \text{ is a closed set of literals or } \exists\, Y, Y \in TP(M) \text{ such that } Y \neq X \text{ and } Y \subseteq X\ \}$$

We denote $\varphi(TP(\{F\}))$ and $\varphi(TS(\{F\}))$ the disjunctive normal forms respectively of $TP(\{F\})$ and $TS(\{F\})$. These two disjunctive normal forms of F are obviously equivalent because $(A \wedge B) \vee A \leftrightarrow A$ and $(A \wedge \neg A \wedge B) \vee C \leftrightarrow C$ where A, B, and C are any formula. TS often reduces the size of the tableau and then is used instead of TP.

3.2 Models as sets of literals

The operator TS applied to the theory $\mathcal{B}$, gives a set $\{T_1, T_2, ..., T_n\}$, where each T_i is a open set of literals of $\mathcal{B}$. The link between the models and the sets T_i is expressed in the next theorem:

Theorem 1 *m is a model for the theory $\mathcal{B}$ if and only if there exists a set T of $TS(\mathcal{B})$ such that $m \models l$ for each literal l of T.*

proof: to apply TS to $\mathcal{B}$ amounts to expressing the theory in a disjunctive normal form and then $\varphi(TS(\mathcal{B}))$ is logically equivalent with the theory $\mathcal{B}$. m is a model of $\mathcal{B}$ if and only

if it is a model of the formula $\varphi(TS(\mathcal{B}))$. But the model m satisfies $\varphi(TS(\mathcal{B}))$ if and only if it satisfies at least one of the disjunction which composes $\varphi(TS(\mathcal{B}))$, i.e. if it satisfies at least one of the formulae $l_1 \wedge l_2 \wedge ... \wedge l_n$ where each l_i belongs to a same T of $TS(\mathcal{B})$. But, $m \models l_1 \wedge l_2 \wedge ... \wedge l_n$ if and only if $m \models l_i$ for i in $\{1, ..., n\}$.

The models of the theory are found with TS, then they have to be compared in order to draw out the minimal ones. But this is not so obvious as we will see in the next example. Let $\mathcal{C}$ be a propositional theory such that $TS(\mathcal{C})$ contains at least the two sets of literals T_1 and T_2:

$$T_1 = \{S(a), S(b)\} \text{ and } T_2 = \{P(a), S(a)\}$$

We would like to compare the models associated with the sets T_1 and T_2. But T_2 neither contains $S(b)$, nor $\neg S(b)$ and it seems difficult to compare them if they do not coincide on all literals other than the literals P, Q, $\neg P$ and $\neg Q$. But

$$P(a) \wedge S(a) \leftrightarrow (P(a) \wedge S(a) \wedge S(b)) \vee (P(a) \wedge S(a) \wedge \neg S(b))$$

By putting $T_2' = \{P(a), S(a), S(b)\}$ and $T_2'' = \{P(a), S(a), \neg S(b)\}$ in place of T_2 in $TS(\mathcal{C})$, we obtain a set logically equivalent and the models associated with T_1, T_2' and T_2'' become comparable. The model from T_1 will be absolutely smaller than the model from T_2 and neither the two models from T_1 and T_2' nor the two models from T_2' and T_2'' will be in relation because they do not coincide on $S(b)$. To draw a conclusion, the set T_2 has been "split" in two sets T_2' and T_2''. This development must be carried on to make the whole sets T_i of $TS(\mathcal{C})$ comparable. We will obtain a tableau equivalent to $TS(\mathcal{B})$ because for some formula F and for some literal l, $F \leftrightarrow (F \wedge l) \vee (F \wedge \neg l)$. A model for the theory will be denoted as a set of literals of this developed tableau.

Notation Let $\mathcal{B}$ be a theory such that $TS(\mathcal{B})$ is not empty. We will denote a model m of $\mathcal{B}$ as an open set of literals that contains one of the set T of $TS(\mathcal{B})$ and which is also maximal for inclusion in the set of literals of $\mathcal{B}$. We say that the model m *stems* from T.

The set of models of the theory is therefore obtained from the elements T of $TS(\mathcal{B})$ that are "completed" in order to become maximal sets of literals such that these remain open. Let us formally define this concept of completion.

3.3 Completion operator

Let $\mathcal{B}$ be a theory containing the predicate symbols P and Q. The set $\mathcal{L}^*(\mathcal{B})$ designates the whole set of atoms built from the language of the theory $\mathcal{B}$ with their opposites. To define it, we introduce the operators $^-$ and * that apply to any set of literals A of the theory: $A^- = \{\neg l, l \in A\}$ and $A^* = A \cup A^-$. Let $\mathcal{L}(\mathcal{B})$ be the set of all atoms built with the language of the theory. We can define $\mathcal{L}^*(\mathcal{B}) = \mathcal{L}(\mathcal{B}) \cup \{\neg l, l \in \mathcal{L}(\mathcal{B})\}$.

Definition 3 *The completion operator C associates a set of literals T of $\mathcal{L}^*(\mathcal{B})$ with a set of sets of literals $C(T)$ such that:*
$C(T) = \{T \cup X, X$ *maximal for inclusion in* $\mathcal{L}^*(\mathcal{B})$ *such that* $T \cup X$ *open*$\}$

With this completion operator and the definition of a model as a set of literals that has been given in the previous section, a model can be considered as an element of a set resulting from the completion of a set T of $TS(\mathcal{B})$.

Theorem 2 *A set of literals m is a model for the theory $\mathcal{B}$ if and only if there exists an element T of $TS(\mathcal{B})$ such that m is an element of $C(T)$.*

4 Minimal models

4.1 Definition

Since a model m has been defined as a maximal open set of litterals, for each literal l of $\mathcal{L}^*(\mathcal{B})$, either l belongs to m or $\neg l$ belongs to m. If we note $\Omega(m)$ the set of literals of m that are not literals with the predicate symbols P, $\neg P$, Q nor $\neg Q$, a condition to be able to compare the two models m and m' is that $\Omega(m) = \Omega(m')$. The other condition is that the set of literals $P(u)$ from m where u is any constant symbol of the interpretation domain is included in the set of literals $P(u)$ from m'. Let us more formally define these ideas: let T be any set of litterals of $\mathcal{L}^*(\mathcal{B})$, we note $\mathcal{P}(\mathrm{T})$ the subset of T such that $\mathcal{P}(\mathrm{T}) = \{P(u) \in T\}$. In the same way, each predicate symbol S of the theory is associated with its $\mathcal{S}$ such that: $\mathcal{S}(T) = \{S(u) \in T\}$. Thus:

$$\Omega(T) = T \backslash (\mathcal{P}(\mathrm{T}) \cup \mathcal{P}^-(\mathrm{T}) \cup \mathcal{Q}(\mathrm{T}) \cup \mathcal{Q}^-(\mathrm{T}))$$

With these notations, the definition of $\leq_{P;Q}$ becomes:

Definition 4 $m \leq_{P;Q} m'$ *iff* $\mathcal{P}(m) \subseteq \mathcal{P}(m')$ *and* $\Omega(m) = \Omega(m')$

It is now easy to complete the sets T of $TS(\mathcal{B})$ to obtain the whole set of models of $\mathcal{B}$, then to compare their respective sets $\Omega(m)$ and $\mathcal{P}(m)$ and to exclude the largest models. We will use a direct method: we will not have to build the whole set of models of $\mathcal{B}$ but the sets of literals T of $TS(\mathcal{B})$ will be completed in a particular way, with respect to a set of literals of $\mathcal{L}^*(\mathcal{B})$ such that the largest models will not belong to these completed sets.

4.2 Completion operator with respect to a set of literals

The definition of the operator C_A, with respect to a set of literals A of $\mathcal{L}^*(\mathcal{B})$ is derived from the definition of the operator C. The elements of the set $C_A(T)$ are open, maximal in $\mathcal{L}^*(\mathcal{B})$ and they contain the set A: they are models that stem from T and which contain A. Therefore, the set $C_A(T)$ is a particular subset of the whole set of models that stem from T.

Definition 5 *C_A is the completion operator with respect to the set of literals A of $\mathcal{L}^*(\mathcal{B})$. It associates a set of literals T of $\mathcal{L}^*(\mathcal{B})$, with a set of sets of literals $C_A(T)$ such that: $C_A(T) = \{A \cup T \cup X, X$ maximal for inclusion in $\mathcal{L}^*(\mathcal{B})$ such that $A \cup T \cup X$ open$\}$*

Property 1 *$C(T) = C_\emptyset(T)$ where $\emptyset$ is the empty set of literals.*

Property 2 $C_A(T) = C_T(A) = C_\emptyset(A \cup T)$

Property 3 *If $A \subseteq B$ then $C_B(T) \subseteq C_A(T)$*

proof: if $C_B(T)$ is not the empty set, each element of $C_B(T)$ is written $T \cup B \cup X$ and is open and maximal for inclusion in $\mathcal{L}^*(\mathcal{B})$. But, since B contains A, this element belongs to $C_A(T)$. In particular, for each set B of literals, $C_B(T) \subseteq C_\emptyset(T)$ which confirms that $C_B(T)$ is a subset of the set of models that stem from T.

Property 4 *$C_A(T) = \{\emptyset\}$ iff $A \cup T$ is closed.*

proof: if $A \cup T$ is closed, then for all X from $\mathcal{L}^*(\mathcal{B})$, $A \cup T \cup X$ is closed and $C_A(T) = \{\emptyset\}$. Let us suppose that $A \cup T$ is opened. We build X choosing for each pair of literals $(l, \neg l)$ where neither l nor $\neg l$ belongs to $A \cup T$, one of the two l or $\neg l$. X does not contain any literal with its opposite, thus it is open and $A \cup T \cup X$ has been built in such a way that it is maximal.

4.3 A first completion

We first note that if m and m' are two models of $\mathcal{B}$ that stem from T and coincide exactly except on a literal $P(u)$ of $\mathcal{L}^*(\mathcal{B})$, ($m$ contains $\neg P(u)$ and m' contains $P(u)$), m' is "absolutely" greater than m. If we complete T with respect to a set A which contains $\neg P(u)$, m' will not be in $C_A(T)$. A first obvious step will consist in completing each T of $\mathcal{L}^*(\mathcal{B})$ with respect to the set N_T^- that contains the set of literals of the form $\neg P$ of $\mathcal{L}^*(\mathcal{B})$ whose opposite does not belong to T.

4.3.1 The set N_T^-

The operator $\mathcal{P}$ has already been introduced in the previous section. If T is some set of literals of $\mathcal{L}^*(\mathcal{B})$, $\mathcal{P}(\mathrm{T}) = \{P(u) \in T\}$. We deduce that: $P(\mathcal{L}^*(\mathcal{B})) = \{P(u) \in \mathcal{L}^*(\mathcal{B})\}$. Each set of literals T is associated with the set N_T composed of the literals $P(u)$ of $\mathcal{L}^*(\mathcal{B})$ such that $P(u)$ doesn't belong to T: $N_T = \mathcal{P}(\mathcal{L}^*(\mathcal{B}))\backslash\mathcal{P}(\mathrm{T})$. Therefore N_T^- is a set of literals $\neg P(u)$ of $\mathcal{L}^*(\mathcal{B})$ such that $P(u)$ doesn't belong to T.

$$N_T^- = (\mathcal{P}(\mathcal{L}^*(\mathcal{B}))\backslash\mathcal{P}(\mathrm{T}))^-$$

Example:
Let $\mathcal{B}$ be a theory on the language composed of the only predicate symbols P, Q and S and of the only constant symbols u and v.

$$\mathcal{L}^*(\mathcal{B}) = \{P(u), \neg P(u), P(v), \neg P(v), Q(u), \neg Q(u), Q(v), \neg Q(v), S(u), \neg S(u), S(v), \neg S(v)\}$$

We suppose that $TS(\mathcal{B}) = \{T, T'\}$ with

$$T = \{\neg S(u), Q(u)\neg Q(v)\} \text{ and } T' = \{S(v), \neg Q(u), P(v)\}$$

$\mathcal{P}(\mathrm{T}) = \emptyset$	$\mathcal{P}(T') = \{P(v)\}$	$P(\mathcal{L}^*(\mathcal{B})) = \{P(u), P(v)\}$
$N_T^- = \{\neg P(u), \neg P(v)\}$	$N_T'^- = \{\neg P(u)\}$	
$\Omega(T) = \{\neg S(u)\}$	$\Omega(T') = \{S(v)\}$	

4.3.2 Theorem

Theorem 3 *Let $\mathcal{B}$ be a theory and T a set of literals that belongs to $TS(\mathcal{B})$. The set of models that stem from T which are minimal for $\leq_{P;Q}$ is a subset of $C_{N_T^-}(T)$.*

proof: let us suppose that there exists a model m that stems from T, minimal for $\leq_{P;Q}$ which does not belong to $C_{N_T^-}(T)$. The model m stems from T, so m belongs to $C(T)$, but not to $C_{N_T^-}(T)$. Then, there exists an element of m that does not belong to N_T^-. From the definition of N_T^-, this element is a $\neg P(u)$ with $P(u)$ not in T. The set of literals that coincides with m everywhere except on $P(u)$ and that contains $\neg P(u)$ is a model that stems from T because it is open, maximal in $\mathcal{L}^*(\mathcal{B})$. The model m' has been built such that m' is absolutely smaller than m. ($\mathcal{P}(m') \subseteq \mathcal{P}(m)$ and $\Omega(m') = \Omega(m)$). The model m is not minimal for $\leq_{P;Q}$.

Definition 6 *T is said to generate minimal models if the set of minimal models for the theory that stems from T is the whole set $C_{N_T^-}(T)$.*

Remarks:

- $\mathcal{P}(m) = \mathcal{P}(\mathrm{T})$ for the elements m of $C_{N_T^-}(T)$.
- If T does not contain any literal $P(u)$ of $\mathcal{L}^*(\mathcal{B})$, (i.e. $\mathcal{P}(\mathrm{T}) = \emptyset$) then obviously, T generates minimal models. Indeed, for each model of $C_{N_T^-}(T)$, $\mathcal{P}(m) = \mathcal{P}(\mathrm{T}) = \emptyset$. No other model of the theory can be absolutely smaller than m. Therefore T generates minimal models.

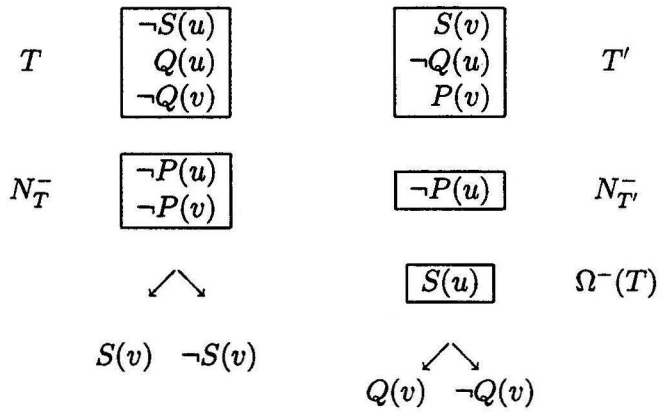
4.4 The set of minimal models

If there exists T and T' in $TS(\mathcal{B})$ such that $\mathcal{P}(\mathrm{T}) \subseteq \mathcal{P}(T')$ and $\Omega(T) \subseteq \Omega(T')$, none of the models of $\mathcal{B}$ that stems from T' can be a minimal model because each element of $C_{N_{T'}^-}(T')$ is absolutely greater than an element of $C_{N_T^-}(T)$ (which contains all the litterals of $\Omega(T')$). If $\Omega(T) \not\subseteq \Omega(T')$ then T' has to be completed with respect to a set of literals A in such a way that none of the models of the completed set coincide with a model that stems from T. Necessarily, A must contain one literal whose opposite belongs to $\Omega(T)$. A will be built taking into account all sets T of $TS(\mathcal{B})$ whose $\mathcal{P}(\mathrm{T})$ is included in $\mathcal{P}(T')$.

Example (following):
$T = \{\neg S(u), Q(u)\neg Q(v)\}$ and $\mathcal{P}(\mathrm{T}) = \emptyset$. From the previous remark, T generates minimal models for the theory.

$T' = \{S(v), \neg Q(u), P(v)\}$ and all the elements of $C_{N_{T'}^-}(T')$ are not minimal models for the theory. For example , the model $m = \{\neg S(u), S(v), \neg P(u), P(v), \neg Q(u), \neg Q(v)\}$, element of $C_{N_{T'}^-}(T')$ is absolutely greater than the model $\{\neg S(u), S(v), \neg P(u), \neg P(v), Q(u), \neg Q(v)\}$ of $C_{N_T^-}(T)$. The only models of $C_{N_{T'}^-}(T')$ that contain $S(u)$ no longer coincide with the models generated by T on the literal $S(u)$. They will not be comparable with the elements

of $C_{N_T^-}(T)$ and therefore are minimal for the theory. The set $C_{N_{T'}^- \cup \{S(u)\}}(T')$ contains minimal models for the theory, or equivalently $T \cup \{S(u)\}$ generates minimal models.



Theorem 4 *Let $\mathcal{B}$ be a theory which contains the predicate symbols P and Q such that $TS(\mathcal{B}) \neq \{\emptyset\}$. Let T be an element of $TS(\mathcal{B})$. We note:*

$$\mathcal{T}_T = \{T' \in TS(\mathcal{B}), \mathcal{P}(T') \subset \mathcal{P}(\mathrm{T})\} = \{T'_1, T'_2, ..., T'_m\}$$

Then for any Y:

$$Y \subseteq \left\{ \bigcup_{i=1}^{m} \Omega^-(T'_i) \right\} \text{ such that } \forall i \in \{1, \ldots, m\}, Y \cap \Omega^-(T'_i) \neq \emptyset$$

$Y \cup T$ either generates $\{\emptyset\}$ or generates minimal models.

That means, if $TS(\mathcal{B}) = \{T_1, T_2, ..., T_n\}$, the set of minimal models of $\mathcal{B}$ is a subset of:

$$\mathcal{M}_{\mathcal{M}} = \bigcup_{i=1}^{n} C_{\{x_1\} \cup \{x_2\} \cup ... \cup \{x_{m_i}\} \cup N_{T_i}^-}(T_i)$$

where $\mathcal{T}_{T_i} = \{T'_1, T'_2, ..., T'_{m_i}\}$ and $\forall j \in \{1, ..., m_i\}, x_j \in \Omega^-(T'_j)$.

proof: let Y be a set of literals which satisfies the conditions of the theorem. If $Y \cup T$ is a closed set of literals, it generates the empty set. If $Y \cup T$ is open, let us prove that it generates minimal models. Let m be some element of $C_{N_{Y \cup T}^-}(Y \cup T)$, then $\mathcal{P}(m) = \mathcal{P}(Y \cup T)$. Since Y is a subset of $\cup_{i=1}^{m} \Omega^-(T'_i)$ where T'_i is in $\mathcal{T}_\mathrm{T}$ and since the sets $\Omega^-(T'_i)$ does not contain any predicate with P, $\mathcal{P}(Y) = \emptyset$ and then $\mathcal{P}(m) = \mathcal{P}(T)$. The only models m' which could be absolutely smaller than m are those such that $\mathcal{P}(m') \subseteq \mathcal{P}(m)$. Those m' necessaraly stem from elements of $\mathcal{T}_\mathrm{T}$. Indeed, if m'' is a model that stems from T'' of $TS(\mathcal{B})$ which does not belong to $\mathcal{T}_\mathrm{T}$, $\mathcal{P}(T'') \subseteq \mathcal{P}(m'')$ because $T'' \subseteq m''$. But if T'' is not in $\mathcal{T}_\mathrm{T}$, $\mathcal{P}(T'') \backslash \mathcal{P}(T)$ is not empty, then since $\mathcal{P}(m) = \mathcal{P}(T)$, $\mathcal{P}(m'') \not\subseteq \mathcal{P}(m)$. The set Y contains at least one element of each set $\Omega^-(T'_i)$, then each model that stems from $T \cup Y$ does at least not coincide on this element with models that stem from the elements of $\mathcal{T}_\mathrm{T}$. So, the smallest models among the models that stem from $Y \cup T$ are minimal models: they are elements of $C_{N_{(Y \cup T)}^-}(Y \cup T)$.

Theorem 5 *Reciprocally, every minimal model is obtained in that way (i.e. the set of minimal models is exactly the set $\mathcal{M}_{\mathcal{M}}$).*

proof: let m be a minimal model for the theory $\mathcal{B}$. There exists a set of literals T of $TS(\mathcal{B})$ such that m belongs to $C_{N_T^-}(T')$ (we have already seen in the previous section that the set of minimal models was a subset of the sets $C_{N_T^-}(T)$). Let T' be an element of $\mathcal{T}_{\mathrm{T}}$, let us prove that there exists at least one literal l in $\Omega^-(T')$ which is also in m. We suppose that there is no literal of $\Omega^-(T')$ in m. If l is in $\Omega^-(T')$ and not in m, then $\neg l$ is in m. This is true for each l of $\Omega^-(T')$, so $(\Omega^-(T'))^- \subseteq m$ which is equivalent with $\Omega(T') \subseteq m$. From T', a model m' can be built such that $\Omega(m') = \Omega(m)$. Besides, since m' stems from T' and T' is in $\mathcal{T}_{\mathrm{T}}$, we have $\mathcal{P}(m') \subset \mathcal{P}(m)$ and then $m' \leq_{P;Q} m$ and m is not minimal for $\leq_{P;Q}$ which is opposite to the hypothesis. We deduce that m contains subsets of literals X_i of $\Omega^-(T_i')$ for each T_i' of $\mathcal{T}_{\mathrm{T}}$. Besides, m belongs to $C_{N_T^-}(T)$, then m belongs to $C_{(\cup_{i=1}^n X_i) \cup N_T^-}(T)$.

These two theorems characterise the set of minimal models of a theory. To compute them, for each T in $TS(\mathcal{B})$, we compare $\mathcal{P}(T)$ with each $\mathcal{P}(T')$ for T' in $TS(\mathcal{B})$ (T' different from T) and we will consider the T' such that $\mathcal{P}(T')$ contains $\mathcal{P}(T)$. Obviously, for such a T', if $\Omega(T')$ contains $\Omega(T)$, T' just have to be pull out because it will not produce minimal models. If $\Omega(T)$ contains a literal whose opposite is in $\Omega(T')$, then the models which stem from T and T' do not coincide on this literal and then, can not be compared. In the opposite case, we associate $\Omega^-(T)$ with T'. We continue this process until all the sets of $TS(\mathcal{B})$ have been compared each others (that represents $n \times (n-1)$ comparisons if n is the number of T in $TS(\mathcal{B})$). After that, each T of $TS(\mathcal{B})$ is associated with some sets of literals. Minimal models which stem from T will be the open sets of literals composed of T, a literal of each set associated with T, N_T^-, and either of the literal l, or of the literal $\neg l$ if neither l nor $\neg l$ already appears in m. But we can improve this algorithm: first, we remove from the sets $\Omega^-(T)$ associated with T', the literals whose opposite is in T'. Indeed, minimal models are opened and do not contain a literal and its opposite. Thus, the sets associated with T' become sets of the form $\Omega^-(T) \backslash \Omega^-(T')$. Moreover, we impose that a model from T' contains at least one literal of each set associated with T' to be minimal. This condition is obviously checked for each set associated with T' which contains another set associated with T': every time a minimal model from T' contains a literal l of a set A associated with T', l belongs to all the associated sets which contain A. Thus, only the smallest associated sets of T' are used to build the minimal models which stem from T'. These inclusions are tested during the minimal models computing.

To give information about minimal models allows us to accurately characterize the minimal conditions when the predicate P appears. The process used to obtain these minimal models clearly showed the way these minimal conditions are induced: in the previous example, we made models that stemmed from T' minimal, by imposing the condition $S(u)$. It is only with this condition that the property P is confirmed for the the constant v. Therefore, we will conclude that $P(x)$ is only true when we have both $x = v$ and $S(u)$. Thus

$$\forall x \, P(x) \leftrightarrow x = v \wedge S(u)$$

Clearly, all the minimal models of the theory agree with this formula.

5 Example

The example of [LEA SOMBE,1990] is now treated with the minimal models. The theory $\mathcal{B}$ is:

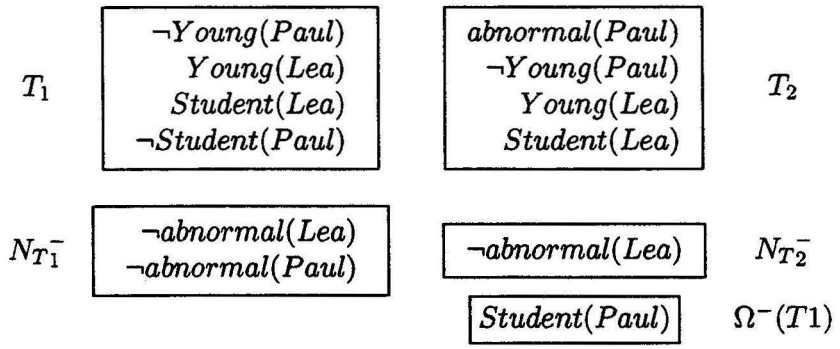
$$\forall x\ Student(x) \wedge \neg abnormal(x) \rightarrow Young(x)$$
$$Student(Lea) \wedge \neg Young(Paul)$$

The predicate *abnormal* is circumsribed and the predicate $Young$ varies. $TS(\mathcal{B}) = \{T_1, T_2, T_3, T_4\}$ with :

T_1	T_2	T_3	T_4
		$abnormal(Lea)$	$abnormal(Lea)$
	$abnormal(Paul)$		$abnormal(Paul)$
$\neg Young(Paul)$	$\neg Young(Paul)$	$\neg Young(Paul)$	$\neg Young(Paul)$
$Young(Lea)$	$Young(Lea)$		
$Student(Lea)$	$Student(Lea)$	$Student(Lea)$	$Student(Lea)$
$\neg Student(Paul)$		$\neg Student(Paul)$	

- T_1 does not contain any predicate *abnormal* so T_1 generates minimal models. $C_{N_{T_1}^-}(T_1)$ only contains minimal models for the theory.
- $\mathcal{T}_{T_2} = \{T_1\}$ and $\Omega^-(T_1) = \{\neg Student(Lea), Student(Paul)\}$. The set of literals $T_2 \cup \{\neg Student(Lea)\}$ is closed, so completing T_2 with respect to $\{\neg Student(Lea)\}$ will give the empty set. $T_2 \cup \{Student(Paul)\}$ only generates minimal models.
- $\mathcal{T}_{T_3} = \{T_1\}$ and $\mathcal{T}_{T_4} = \{T_1, T_2, T_3\}$. To be minimal, the models that stem from T_3 and T_4 have to be completed with respect to a set which contains the element of $\Omega^-(T_2)$ which is $\neg Student(Lea)$. But, T_3 and T_4 contain the predicate $Student(Lea)$ then T_3 and T_4 do not generate minimal models.

There are two minimal models for the theory and, using the same diagram as in example 1:



The predicate *abnormal* only appears if there is $Student(Paul)$ and only concerns $Paul$. Thus:

$$\forall x\ abnormal(x) \leftrightarrow x = Paul \wedge Student(Paul)$$

This formula is true for the minimal models of $\mathcal{B}$. Wherever the predicate *abnormal* appears in the theory, it could be substituted with the right part of the formula which is exactly the instance of ϕ_P in the circumscription diagram. The result found in the first section is confirmed:

$$\forall x\ Student(x) \wedge x \neq Paul \rightarrow Young(x)$$

6 Conclusion

We have presented here a method to compute minimal models for a propositional theory, for the particular relation $\leq_{P;Q}$ associated with the circumscription with fixed literals and with predicates that are allowed to vary. We have also seen with an example how to characterise the exceptions and then make deductions from these minimal models. It would be interesting to generalize these deductions and to prove that they correspond with "right" instances for φ_P and ϕ_Q. This could be done in propositional logic and perharps in universal first order logic but with more difficulty in any first order logic because syntactical and semantical entailment do not always coincide. It would be also interesting to add rules to reduce the analytic tableau, using the properties of the predicate Q that is allowed to vary. For example, if the formula $Q(u) \vee P(v)$ belongs to the theory and if the branch developed from $Q(u)$ is open, there is no need to develop a branch from $P(v)$ because for each model of this branch there will a model from the other branch absolutely smaller. Other properties can be found to exclude directly some models that are obviously not minimal and to reduce to a minimum the sets T of $TS(\mathcal{B})$ that will completed.

References

1. Bossu G., Siegel P.,"*Saturation, Nonmonotonic Reasoning and the Closed World Assumption*", *Artificial Intelligence* 25, 1985, pp 13-63.
2. Besnard P.,"*An Introduction to Default Logic*", Springer-Verlag, Heidelberg, 1989, pp 131-160.
3. Besnard P., Siegel P.,"*The Preferential-Models Approach to Non-monotonic Logics*", in: *Non-Standard Logics for Automated Reasonning*, Smets Mandani Dubois Prade (Eds), Academic Press, 1988, pp137-161.
4. Brewka G,"*Non-monotonic Reasonning: Logical Fundations of Commonsense*", Cambridge Tracts in Theorical Computer Science 12, Cambridge University Press, 1991.
5. De Kleer J., Konolige K.,"*Eleminating the Fixed Predicates from a Circumscription*", *Artificial Intelligence* 39, 1990, pp 75-94.
6. Fitting M,"*First Order Logic and Automated Theorem Proving*", Springer-Verlag, N.Y.,1990.
7. Hintikka J.,"*Model Minimization an Alternative to Circumscription*", *Journal of Automated Reasoning* 4, 1988, pp 1-13.
8. Lifschitz V.,"*Computing Circumscription*", *Proceedings of IJCAI*, Los Angeles, 1985, pp 121-127.
9. McCarthy J.,"*Circumscription-A Form of Non-Monotonic Reasoning*", *Artificial Intelligence* 13, 1980, pp 49-73.
10. McCarthy J.,"*Applications of Circumscription to Formalizing Commonsence Knowledge*", *Artificial Intelligence* 28, 1986, pp 89-118.
11. Ginsberg M.L.,"*A circumscriptive Theorem Prover*", *Artificial Intelligence* 39, 1989, pp 209-230.
12. Moinard Y.,"*Raisonnement Non-Monotone: Contribution à l' Etude de la Circonscription* ", *thèse*, Université de Rennes I, 1988.
13. Olivetti N.,"*Tableaux and Sequent Calculus for Minimal Entailment*", *Journal of Automated Reasonning* 9, 1992, pp 99-139.
14. Perlis D.,Minker J.,"*Completeness Results for Circumscription*", *Artificial Intelligence* 28, 1986, pp 29-42.
15. Przimunsinsky T.C.,"*An Algorithm to Compute Circumscription*", *Artificial Intelligence* 38, 1989, pp 49-73.
16. Shoham Y.,"*A Semantical Approach to Nonmonotonic Logics*", *Proceedings of Logics*

in Computer Science, Ithaca,N.Y., 1987, pp 227-250.
17. Sombe Lea,"*Reasonning under Incomplete Information in Artificial Intelligence*", Wiley, 1990.
18. Schwind C.,"*A Tableau-Based Theorem Prover for a Decidable Subset of Default Logic*", *Proceedings of the 10th Conference on Automated Deduction*, Spinger Verlag, 1990, pp 541-546.
19. Schwind C., Risch V.,"*A Tableau-Based Characterisation for Default Logic*", *Proceedings of the European Conference on Symbolic and Quantitative Approaches for Uncertainty*, Marseille, Lecture Notes in Computer Science 548, Springer Verlag, 1991, pp 310-317.
20. Smullyan R.M.,"*First Order Logic*", Springer-Verlag, Berlin,1968.

A Tableau Prover for Domain Minimization*

Sven Lorenz
IBM Germany Science Center · WZH IWBS
Vangerowstr. 18 · 6900 Heidelberg · Germany
email: lorenz@dhdibm1.bitnet

1 Introduction

This article presents a semantically sound tableau-based proof-procedure for a class of nonmonotonic logics that can be characterized by the term *domain minimization*. Domain minimization is a form of nonmonotonic reasoning that roughly speaking expresses the conjecture "that the 'known' entitities are all there are" [McC80]. Two forms of domain minimization have been described in the literature: McCarthy's *domain circumscription* with *minimal entailment* as its semantic counterpart [McC80], and Hintikka's *mini-consequence*, a proof theoretic notion based on semantic tableaux [Hin88].

Domain circumscription is easily expressed semantically: instead of all models of a set of first-order sentences Φ only those models with minimal domains are considered wrt. entailment. A sentence φ is a minimal entailment of Φ iff it is true in all domain minimal models of Φ. As a proof theory for minimal entailment McCarthy suggested a *domain-circumscription schema*, i.e. an axiom schema that replaces the original set of axioms Φ and whose models are the minimal models of Φ[1]. Suitable instantiations of this schema yield the minimal consequences of the circumscribed set of axioms. However, for circumscriptive theorem provers it has been a major problem to find just the right instantiations that prove a goal formula or result in non-trivial consequences [Gin89; Prz89] [2].

Mini-consequence, Hintikka's alternative to domain circumscription, is a syntactically defined nonmonotonic consequence relation for a first-order language without function symbols. It can be regarded as a procedure to find suitable *domain-closure axioms* for a given set of axioms and then prove theorems from these domain-closure axioms and the axiom set. The basic idea is to explicitly construct (partial) models using semantic tableaux. Thus, one is interested in the complete **open** branches of a tableau, since those can be regarded as representing partial models of the set of axioms for which the tableau has been constructed. By a slight change in the tableau rule for the existential quantifier one can obtain in a certain sense "domain minimal" partial models, i.e. complete open branches that satisfy certain minimality conditions. These "minimal" branches are then used to refute the negation of the goal formula in the standard way, resulting in a nonmonotonic consequence relation.

*The author would like to thank Stephan Bayerl, Torsten Schaub, and an anonymous referee for valuable remarks on earlier versions of this article.

[1] Domain circumscription in its schema version is sound wrt. minimal entailment. Completeness is obtained for restricted classes of theories or in the second-order version of circumscription. See [EM87] for more details.

[2] The cited proof methods have actually been designed for predicate circumscription, so one might suspect that the criticism only applies to this version of circumscription. However, to my knowledge a special domain-circumscriptive prover has never been proposed. As I see it applying the techniques of [Gin89; Prz89] to domain circumscription would lead to similar problems with finding suitable instantiations of the predicate parameter in the schema as in the case of predicate circumscription.

Hintikka also sketches a semantics for mini-consequence, though only for a very restricted language with exactly one two-place predicate and no constant or function symbols. The relation to circumscription[3] is only illustrated by some examples.

The aim of this article is three-fold: First we will illustrate that mini-consequence, although proof-theoretically appealing, is in its current form semantically not well-behaved: logically equivalent sentences have different mini-consequences depending on their syntactic form. Thus a general semantics for mini-consequence would have to take into account the syntax of the sentences which is not desirable. This poses two questions: How can mini-consequence be modified to overcome this problem and what is an intuitive semantics for this modified calculus?

Second, we will suggest a modification of mini-consequence that results in a stronger consequence relation and at the same time avoids some of the problems of mini-consequence. It will turn out that this stronger notion of mini-consequence has a very simple and intuitive minimal model semantics, called *variable minimal entailment*, which is itself a strengthening of the semantics of domain circumscription. Hence, we also clarify the relation of mini-consequence to domain circumscription: in their strong form they are equivalent. Thus, the tableau-based calculus for variable minimal entailment does represent an alternative to domain circumscription.

The third part of the paper is devoted to a discussion of MINITAB, a tableau theorem prover that implements variable minimal entailment for an order-sorted predicate logic. Its basic calculus mostly uses standard implementation techniques for tableau theorem provers, so we will concentrate on those issues directly related to the construction of MINImal TABleaux.

2 Hintikka's Mini-Consequence

Hintikka originally defined mini-consequence on the basis of Beth's semantic tableaux. We will transfer his definitions to an unsigned Smullyan-style tableau calculus because it lends itself more readily to an implementation and can be better illustrated graphically. We assume some familiarity with semantic tableaux and only give a very short introduction. A more comprehensive account of tableau calculi is given e.g. in [Fit90].

Figure 1 shows Smullyan's *uniform notation* of first-order formulas and the associated tableau expansion rules.

A *tableau* is a tree structure with nodes labeled by formulas. A path from the root to a leaf node is called a *branch*. The *initial tableau* for a finite set of the axioms Φ consists of exactly one branch containing the formulas of Φ. A branch B of a tableau T for Φ may be expanded according to one of the rules in Figure 1, resulting in a new tableau T' for Φ. A branch B is called *closed* iff a formula and its negation occur on B; otherwise it is called *open*. A tableau is called *closed* iff all of its branches are closed. A branch B is called *complete* iff it is closed or no more rules can be applied to it, i.e. each α, β, δ- and $\neg\neg$-formula on B has been expanded once and each γ-formula has been expanded once for each constant on B. A tableau is called *complete* if all of its branches are complete. A sentence φ is called a *tableau consequence* of a set of axioms Φ iff there is a closed tableau for $\Phi \cup \{\neg\varphi\}$.

The central theorem of classical tableau calculus is its soundness and completeness wrt. classical model theory.

Theorem 2.1 (Smullyan) *Let Φ be a set of first-order sentences without function symbols. A sentence φ is a tableau consequence of Φ iff φ is true in all models of Φ.*

[3]Interestingly Hintikka compares his approach to predicate circumscription and not to domain circumscription which is conceptually closer to the intuitions of mini-consequence.

α-formulas	α_1	α_2	β-formulas	β_1	β_2
$\varphi \wedge \psi$	φ	ψ	$\varphi \vee \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$	$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \rightarrow \psi)$	φ	$\neg\psi$	$\varphi \rightarrow \psi$	$\neg\varphi$	ψ

γ-formulas	$\gamma[\mathbf{x}]$		δ-formulas	$\delta[\mathbf{x}]$	
$\forall x.\varphi[\mathbf{x}]$	$\varphi[\mathbf{x}]$		$\exists x.\varphi[\mathbf{x}]$	$\varphi[\mathbf{x}]$	
$\neg\exists x.\varphi[\mathbf{x}]$	$\neg\varphi[\mathbf{x}]$		$\neg\forall x.\varphi[\mathbf{x}]$	$\neg\varphi[\mathbf{x}]$	

α-rule	β-rule	γ-rule	δ-rule	$\neg\neg$-rule
α	β	γ	δ	$\neg\neg\varphi$
α_1	$\beta_1 \mid \beta_2$	$\gamma[a]$	$\delta[sk]$	φ
α_2		for an arbitrary constant a	for a new Skolem constant sk	for any formula φ

Figure 1: Smullyan's uniform notation and the corresponding tableau rules

Now, Hintikka's idea can be described roughly as follows: Complete open branches in a semantic tableau can be considered as representing sets of models of the axioms for which the tableau has been constructed. In order to minimize the domains of these models Hintikka suggests a simple modification of the δ-rule. The standard rule introduces a new (i.e. not yet used in the tableau construction) Skolem constant for the existentially quantified variable. Potentially smaller models could result from reusing "old" constants – constants that already occur on the current branch. Thus, alternative branches with reused constants are constructed in parallel with standard branches. The complete open branches that reused as many constants as possible instead of introducing new ones represent in a certain sense "minimal" models. This minimality is "recorded" by adding a *domain-closure axiom (DCA)* $\forall x.x \doteq a_1 \vee .. \vee x \doteq a_n$, the a_i being the constants occurring on the branch, to each "minimal" branch. A sentence φ is a *mini-consequence* of a set of axioms Φ iff the minimal branches extended by their domain-closure axiom and the negation of φ can be closed in the standard way.

$$\frac{\delta}{\delta[sk] \mid \delta[a_1] \mid \ldots \mid \delta[a_n]}$$

for a Skolem constant sk and
all constants $a_1, .., a_n$ on the current branch

Figure 2: The δ^*-rule

Let us look at this in more detail. The δ^*-rule that replaces the original δ-rule is shown in Figure 2. Instead of only continuing the branch with the δ-formula instantiated by a new (Skolem-)constant, parallel branches are started that instantiate all the "old" constants already occurring on the branch. These parallel branches are then expanded in the same way as the regular branch. In a sense this corresponds to a check, whether the existence condition of the δ-formula could be fulfilled by a "known" element of the domain, i.e. an element already named by one of the "old" constants[4].

[4]Some remarks concerning my somewhat loose terminology: I will distinguish several sets of constant symbols on a particular tableau branch. The constants that already occur in the axioms for which the

Figure 3 shows a complete tableau for $\Phi_0 = \{\exists x.q(x), q(a) \rightarrow \exists x.x \neq a\}$. The complex formulas in the tableau are numbered. The numbers on the edges indicate the expansion of which formula produced the edges. The branches containing only solid edges are called *primary branches*. They make up the classical part of the tableau. Dashed edges are produced by δ^*-applications for those branches that reuse an old constant. A closed branch is indicated by double lines underneath a terminal node.

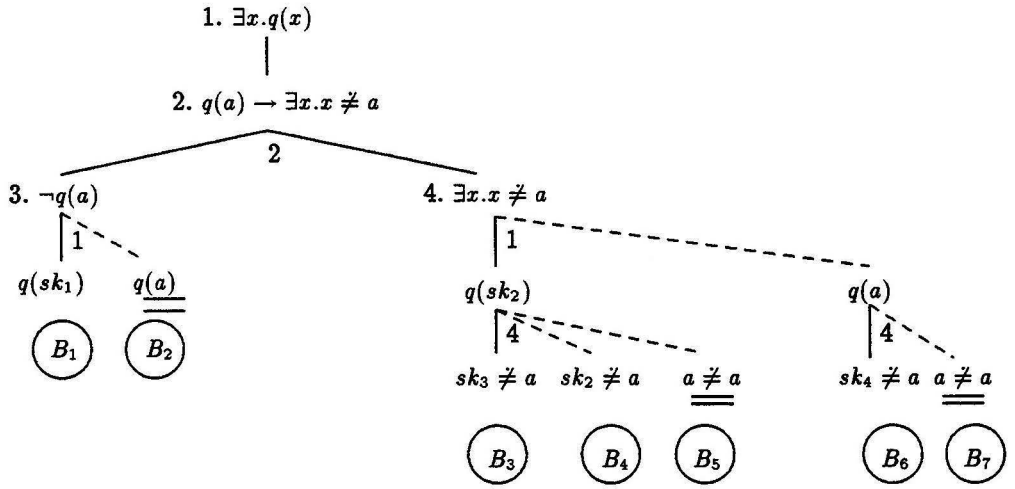


Figure 3: A complete tableau for $\Phi_0 = \{\exists x.q(x), q(a) \rightarrow \exists x.x \neq a\}$

A complete branch A, that reused an old constant but otherwise has been extended in the same way as a complete branch B which introduced a new Skolem constant instead, is called a *subbranch* of B. In Figure 3, for instance, B_2 is a subbranch of B_1 and B_6 is a subbranch of B_3. However, B_6 is **not** a subbranch of B_4, because wrt. the δ^*-expansion of formula 4, B_4 and B_6 were treated differently: on B_4 the constant sk_2 was reused, but on B_6 a new constant (sk_4) was introduced.

If A is a subbranch of B then A contains at least one Skolem constant less than B. So A identifies two constants, say a_1 and a_2, that B keeps separate. Provided A is an open branch, i.e. the reuse of old constants did not lead to an inconsistency, then so is B and both have "almost the same set of models", because the same formulas occur on both branches (modulo the different constants). Only those models of B that interpret a_1 and a_2 differently, and thus might require a bigger domain, cannot be models of A. So in a sense A represents potentially smaller models than B[5].

The subbranch relation constitutes a partial ordering on the branches of a complete tableau. The minimal, open branches according to this ordering we call *H-minimal branches* (H for Hintikka). On H-minimal branches as many δ-introduced elements are identified with elements named by "old" constants as is consistent (because the branch is open) with the given axioms. In order to exclude unnecessarily large models and thus manifest the desired minimality of the models of an H-minimal branch B^* all constants

tableau is constructed are called *original constants*. Thus, the original constants occur on every branch of a tableau. All other constants must have been introduced by an application of the δ^*-rule and are called *Skolem constants*. With respect to a particular δ^*-expansion on a branch I will call constants *old* if they already occur on the branch currently expanded. The *new constant* in this situation is the Skolem constant introduced in this step.

[5]Only "potentially" because the models might contain many other domain elements besides those interpreting the constants on the branches. Domain closure axioms will exclude those unnecessarily large models (see below).

occurring on B^*, say $\{a_1, .., a_n\}$, are used to define a *domain closure axiom* for B^*:

$$DCA(B^*) := \forall x.x \doteq a_1 \vee .. \vee x \doteq a_n$$

A sentence φ is said to be a *mini-consequence* of a set of axioms Φ iff all H-minimal branches of a complete tableau of Φ extended by their corresponding DCAs and $\neg\varphi$ can be expanded to a closed tableau.

A mini-consequence of the axioms Φ_0 from above is, for instance, that there are at most two elements in the universe: The sentence $\varphi = \exists y \forall x.\, x \doteq y \vee x \doteq a$ is a mini-consequence of $\Phi_0 = \{\exists x.q(x), q(a) \rightarrow \exists x.x \neq a\}$, because the H-minimal branches B_1, B_4 and B_6 extended by their DCAs and $\neg\varphi$ can be closed (see Figure 4).

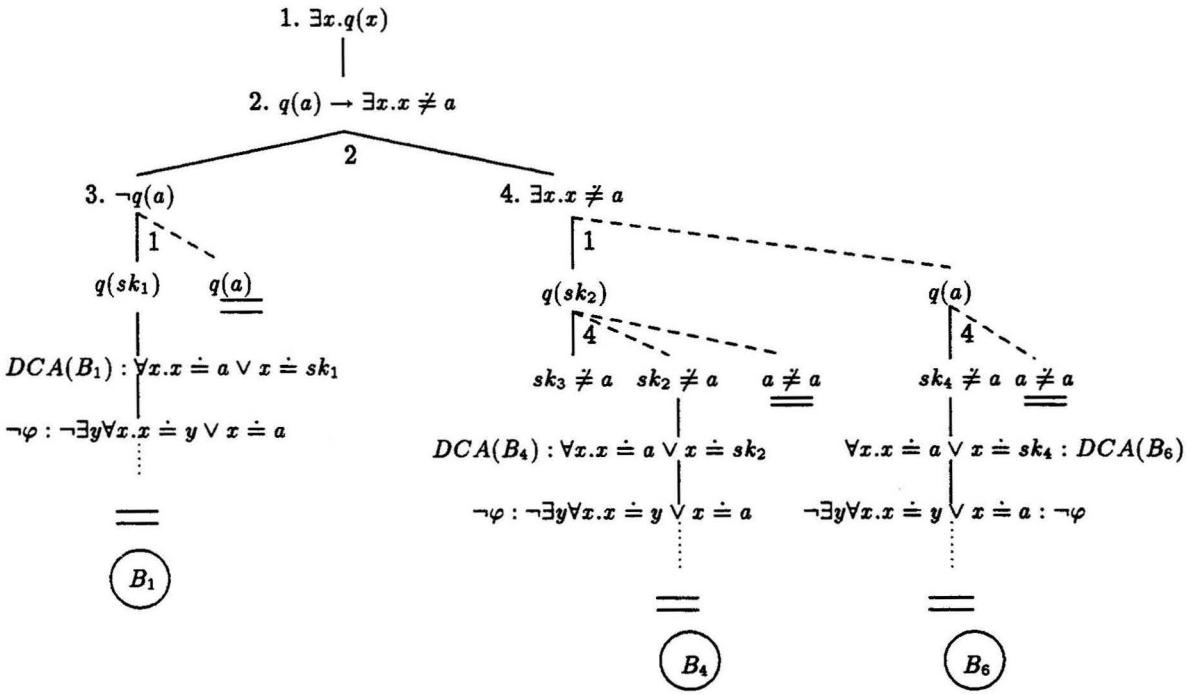


Figure 4: An example for mini-consequence

However, a simple example shows that mini-consequence depends on the syntactic form of the axioms. Figure 5 shows two complete tableaux, one for $\Phi_1 = \{p(a) \vee [\exists x.x \neq a \wedge p(a)]\}$ and one for the logically equivalent $\Phi_2 = \{p(a)\}$.

In the first tableau there are two H-minimal branches, B_1 and B_2, since they both don't have open subbranches. The second tableau trivially has just one H-minimal branch B_1', whose DCA is $\forall x.x \doteq a$. Hence, the sentence $\forall x.x \doteq a$ is a mini-consequence of Φ_2. It is **not** a mini-consequence of Φ_1, however, since branch B_2 in the tableau of Φ_1 contains $\exists x.x \neq a$.

The problem is that for mini-consequence each primary open branch of a complete tableau, together with its open subbranches, represents one alternative set of models and minimization takes place only **within** those alternatives, but not **across** alternatives. Since alternatives are constructed syntactically there is no way to exclude alternatives that are subsumed by others as, for instance, branch B_2 by B_1 in the tableau of Φ_1.

3 Sk-Minimal Consequences

It turns out that by slightly strengthening the notion of mini-consequence we can obtain a proof theory that realizes the comparison of domains across alternatives. As a result the

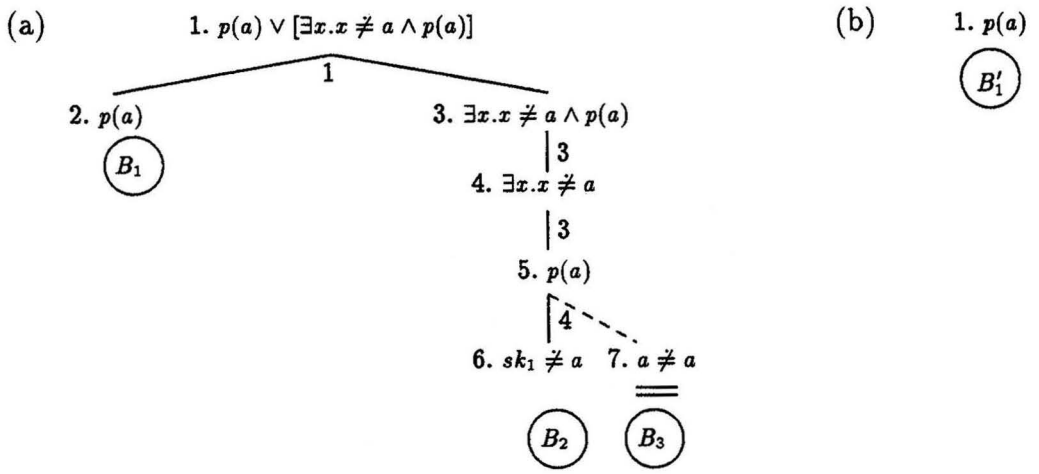


Figure 5: Mini-consequence depends on the syntactic form of the axioms

consequences do not depend on the syntactic form of the axioms. Moreover this modified calculus will be sound and complete wrt. a strong version of minimal entailment.

Intuitively the across-alternatives (i.e. across-branches) comparison rests on the following observation: Provided that the "original" constants, i.e. those constants that already occur in the axiom set, are interpreted identically, the smallest possible size of a branch model only depends on the number of Skolem constants that have to be interpreted differently. The more of these disjunct Skolem constants occur on a branch, the bigger the models of this branch have to be. Hence, when constructing a complete tableau using the δ^*-rule we should compare the branches that allow the identical interpretation of the original constants and select those branches which introduce a minimal number of Skolem constants.

We call two complete, open branches B_1, B_2 *Sk-comparable* iff it is **not** the case that there are constants a, b such that $a \doteq b$ is on B_1 and $a \not\doteq b$ is on B_2[6]. For Sk-comparable branches there always exist models which interpret the original constants identically.

A complete open branch B_1 is called an *Sk-subbranch* of a complete open branch B_2 iff (i) B_1 and B_2 are Sk-comparable, and (ii) B_1 introduces less Skolem constants than B_2. A complete, open branch B^* is called *Sk-minimal* iff there is no Sk-subbranch of B^*.

Sk-minimal consequence is defined completely analogous to mini-consequence. A sentence φ is an *Sk-minimal consequence* of a set of axioms Φ ($\Phi \vdash_{Sk-min} \varphi$) iff all Sk-minimal branches of a complete tableau of Φ extended by their corresponding DCAs and $\neg\varphi$ can be expanded to a closed tableau.

A simple example (Figure 6) illustrates the notion of Sk-minimal branches, Sk-minimal consequence, and the differences to mini-consequence. There are two complete open branches in the tableau of

$$\Phi_3 = \{p(a), p(a) \wedge \neg[\exists x.x \neq a] \rightarrow q(a)\},$$

namely B_2 and B_4. The two branches are Sk-comparable and B_4 introduces less Skolem constants. Hence, B_4 is the only Sk-minimal branch. $q(a)$ is therefore an Sk-minimal consequence of Φ_3. However, both, B_2 and B_4, are H-minimal branches, so $q(a)$ is not a mini-consequence of Φ_3. The following proposition states that Sk-minimal consequence is indeed stronger than mini-consequence.

Proposition 3.1 *Every Sk-minimal branch is H-minimal.*

[6]We assume that every set of sentences contains some axiomatization of equality, e.g. the axiom of reflexivity and replacement axioms for all predicates.

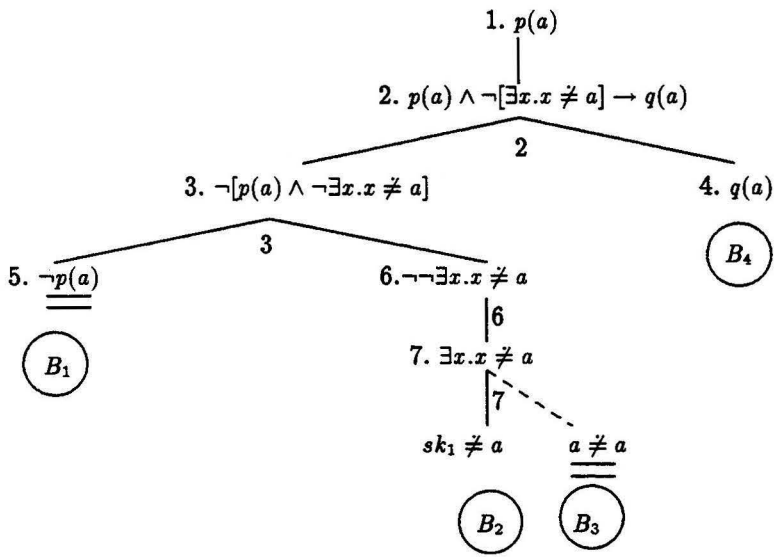


Figure 6: An illustration of Sk-minimal consequence

Note that the problematic example for mini-consequence in Figure 5 on the previous page is adequately treated now. The H-minimal branch B_1 in Figure 5(a) is not Sk-minimal. Thus, Sk-minimal consequence does not run into the same problems as mini-consequence. However, we still have to prove that Sk-minimal consequence does have an intuitive model-theoretic semantics.

4 Minimal and Variable Minimal Entailment

As pointed out in the introduction minimal entailment is the semantic counterpart to domain circumscription. It defines a partial order on the classical models of a set of sentences in the following way: Given two models $M_1 = (D_1, I_1), M_2 = (D_2, I_2)$ of a set of sentences Φ with domains D_i and interpretation functions I_i, M_1 is called a *submodel* of M_2 ($M_1 < M_2$) iff

1. $D_1 \subset D_2$,
2. $I_1(c) = I_2(c)$ for all constants c in Φ, and
3. $I_1(p) = I_2(p)$ restricted on D_1^n for each n-place predicate p.

M is a *(domain-)minimal model* of Φ iff there is no $M' \models \Phi$ with $M' < M$. A sentence φ is a minimal entailment of Φ ($\Phi \models_{min} \varphi$) iff it is true in all minimal models of Φ.

Figure 7 shows two finite models of Φ_3, the axiom set of our last example in the previous section. They are not comparable wrt. "<" because they do not agree on the extension of q. Hence, both models are minimal and Φ_3 does not minimally entail $q(a)$. Obviously minimal entailment is weaker than Sk-minimal consequence. We will see below that in most cases minimal entailment is too weak to conclude anything interesting that goes beyond the classical entailments of a set of sentences.

There are theories without any minimal models as well as there are theories with multiple minimal models. For certain classes of theories minimal models always exist, for instance, for theories with finite models or universal theories[7]. An interesting class are

[7]For a detailed discussion of properties of domain circumscription and minimal entailment see [EM87].

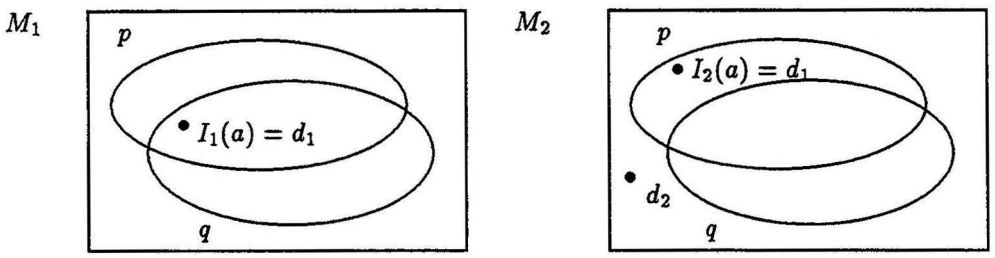


Figure 7: Two minimal models of $\Phi_3 = \{p(a), p(a) \wedge \neg[\exists x.x \neq a] \rightarrow q(a)\}$

so-called *well-founded* theories: a theory Φ is well-founded iff for each model of Φ there is a minimal submodel. Again universal theories are well-founded for minimal entailment.

A well-known result for *predicate* circumscription, which minimizes the extension of a predicate instead of the entire domain [McC80], says that by circumscribing a predicate of a well-founded theory no new information about the extensions of other predicates can be inferred. The corresponding result can also be established for domain circumscription/minimal entailment as is shown in the following proposition.

Proposition 4.1 *Let Φ be a set of axioms whose corresponding theory is well-founded, p an n-place predicate symbol occurring in Φ. Then the following is true:*

1. $\Phi \models_{min} p(t_1, .., t_n)$ *iff* $\Phi \models p(t_1, .., t_n)$ *and*
2. $\Phi \models_{min} \neg p(t_1, .., t_n)$ *iff* $\Phi \models \neg p(t_1, .., t_n)$ *for ground terms* $t_1, .., t_n$.

Thus, a set of axioms corresponding to a well-founded theory minimally entails exactly those ground literals that it classically entails. Note that in this case also no new ground instances of (in-)equalities are minimally entailed. So minimal entailment is a rather weak nonmonotonic entailment relation. In the case of predicate circumscription McCarthy suggested to let the extensions of certain predicates vary *(variable circumscription)* in order to make more models comparable and strengthen the resulting entailment relation [McC86]. For domain circumscription this has never been discussed, though it turns out that this is just the right thing to do in order to overcome the weaknesses illustrated above. At the same time we also obtain a semantics for Sk-minimal consequence.

The transition from minimal entailment to variable minimal entailment is simple: we just have to drop the third condition in the definition of a submodel above, thereby allowing the extensions of all predicates to vary.

Thus, we call M_1 a *variable submodel* of M_2 ($M_1 \ll M_2$) iff

1. $D_1 \subset D_2$ and
2. $I_1(c) = I_2(c)$ for all constants c in Φ.

M is called a *variable (domain-)minimal model* of Φ iff there is no $M' \models \Phi$ with $M' \ll M$. A sentence φ is a *variable minimal entailment* of a set of sentences Φ ($\Phi \models_{vmin} \varphi$) iff φ is true in every variable minimal model of Φ.

Going back to the models of Figure 7 we observe that M_1 is the only variable minimal model of the theory Φ_3 and thus $q(a)$ is a variable minimal entailment of Φ_3. So Sk-minimal consequence and variable minimal entailment agree in this case. We can actually prove that for certain theories Sk-minimal consequence is sound and complete wrt. variable minimal entailment.

Theorem 4.1 *Let Φ be a theory that is well-founded for variable minimal entailment and whose variable minimal models are finite.*

A sentence φ is an Sk-minimal consequence of Φ iff φ is a variable minimal entailment of Φ.

By strengthening the semantics of domain circumscription and also strengthening mini-consequence we were able to provide a minimal model semantics in the spirit of circumscription to an elegant tableau-based proof theory for domain minimization. We will now try to illustrate how Sk-minimal consequence can effectively be implemented by extending standard tableau theorem proving techniques.

5 MiniTab: Domain Minimization Implemented

Sk-minimal consequence has been implemented as an order-sorted, free-variable tableau calculus with unification in a fashion similar to Fitting's implementation described in [Fit90]. Equality is integrated as a special check for branch closure using an explicit construction of equivalence and disjointness classes [BH92]. Since for the classical part of the calculus we mostly used standard implementation techniques, we will discuss only those issues relevant to constructing Sk-minimal branches.

Why order-sorted domain minimization? A strong limitation on our language was the absence of functions. The classical tableau calculus can be easily extended to cover a language that allows function symbols. The only change is in the γ-rule which is allowed to instantiate an arbitrary ground term instead of only a constant. However, it is immediately seen that our modified calculus cannot be extended to cover a language with function symbols in a straight-forward way. Every ground term of the language would have to be "tested" in the δ^*-rule. Hence, with only one constant and one function symbol an application of the δ^*-rule would produce an infinitely branching tableau. The only way to handle functions is to eliminate n-place functions by introducing $n+1$-place predicates (plus the corresponding axioms for totality and functionality) and then apply the calculus to the function-free theory. However, one has to be aware of the fact that by coding functions as predicates and letting predicate extensions vary, one compares not only models that have different extensions for predicates but also different function values for the same arguments. This might not always be feasible.

On the other hand the calculus can easily be extended to an order-sorted language in which it can be used to minimize certain subsets of the domain. For simplicity we will assume that the part of the domain that is minimized is denoted by one particular sort symbol, S_{min}, the *sort to be minimized*. In an order-sorted language both restrictions on axiom sets – the absence of functions and the existence of finite submodels – can be relaxed. One has to avoid (or else code as predicates) only those functions whose range lies within S_{min}. Similarly the requirement that for each model of the axioms a finite variable submodel exists can be relaxed to the requirement that submodels exist that need to be finite only in the minimized part of the domain.

We will not go into details how the classical tableau calculus must be extended to cover an order-sorted language. Basically the only thing that has to be done in a free-variable tableau implementation is to replace standard unification by order-sorted unification.

Free-Variable Tableaux. One of the problems for tableau theorem provers is the indeterminism of the γ-rule. The selection of the "right" constant or ground term is difficult. Usually one has to fall back on some predefined or heuristically controlled enumeration of

these ground terms. A better solution is to delay this selection until a promising instantiation, for instance one that closes a branch, can be chosen. Instead of a ground term the γ-rule introduces a free variable whose value might be determined by unification when a branch is closed.

Unfortunately, due to this delayed selection of ground terms in the γ-rule, the δ-rule has to be modified, too. The introduction of a new Skolem constant does not guarantee any longer that this constant is really "new". Fitting proposes to use Skolem *functions* instead, that take as arguments all previously introduced free variables on the branch [Fit90]. Hähnle and Schmitt showed that it is not necessary to consider **all** free variables on the branch but only those that occur free in the δ-formula to be expanded [HS92]. Still it might be the case that functions whose range is within S_{min} have to be introduced. In the previous subsection it was argued that this is not acceptable. MINITAB offers two ways out of this dilemma:

1. For some axiom sets it can be shown before the actual tableau construction that no δ-expansion will ever introduce a Skolem function. For reasons of well-sortedness this is even more often the case in an order-sorted setting. Hence, when pre-processing an axiom set MINITAB checks whether it can avoid introducing Skolem functions whose range is within S_{min} . If so, then MINITAB uses the standard free-variable γ-rule.

2. If Skolem functions cannot be ruled out, MINITAB has to fall back on an enumeration strategy. At first sight this seems to be awfully inefficient, but in the order-sorted case again the consequences are not quite as drastic.

   The order-sorted free-variable γ-rule is split into two cases: only for variables of the sort S_{min} (and subsorts) it instantiates ground terms, for variables outside S_{min} it introduces free variables. Since S_{min} is restricted to constants and the minimized part of the domain has finite models, there will always be only a finite number of constants to be instantiated.

   The order-sorted free-variable δ^*-rule is also split: if the existentially quantified variable is of the sort S_{min} (or a subsort), then a Skolem *constant* can be chosen. Otherwise a skolem *function* is introduced that takes as arguments the free variables occurring in the δ-formula.

By this mixed strategy MINITAB can benefit from the efficiency gains of a unification-based tableau system, yet avoid functions of sort S_{min} .

Generating Sk-minimal Branches. Besides the γ-rule there are three more major sources of indeterminism in a free-variable tableau-construction: (1) the selection of the next branch to be expanded, (2) the selection of the formula on the branch, and (3) the selection of a substitution that closes a branch (there might be more than one pair of complementary formulas).

The first two kinds of indeterminism are resolved by MINITAB in a straight forward way. Basically MINITAB employs a *depth first expansion* strategy with a limit on the number of γ-expansions per γ-formula in order to assure termination (γ-*limit*). The second indeterminism is resolved by defining an order on formula types, e.g. $\neg\neg \prec \alpha \prec \beta \prec \delta \prec \gamma$, and an order on the formulas of one type, e.g. the order of occurrence on the branch. The selection procedure has to be *fair* in the sense that each formula on a branch is eventually expanded. Using the two orders above and treating the γ-formulas on a branch as a queue yields a fair selection procedure [Sch87].

The interesting case for MINITAB is the resolution of the third type of indeterminism. The standard strategy is *backtracking*: the prover executes one variable substitution for a branch closure at a time, with the option to backtrack to other closing substitutions if later branches cannot be closed. In the case of a complete failure to close a branch (either because the γ-limit was chosen too small or because it is a satisfiable branch) this results in backtracking until the very initial state of the tableau construction.

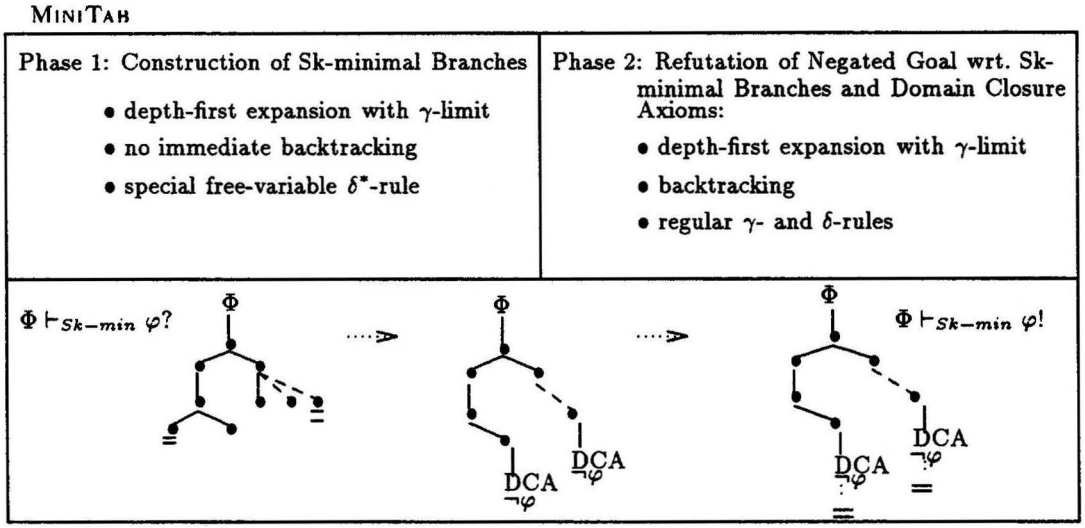


Figure 8: A schematic overview of MINITAB

Given this overall strategy of the tableau construction the first phase of MINITAB, the generation of Sk-minimal branches, runs into problems. The strategy is well-suited for refutation provers that try to construct closed tableaux. Hence, it can be used in the second phase of MINITAB, the refutation of the Sk-minimal branches extended by their domain closure axioms and the negated goal formula. But backtracking has to be suppressed during the construction of Sk-minimal branches. Otherwise no open branch could ever be reached and checked for SK-minimality. However, backtracking might be initiated in the second phase and then backtracking into the first phase can occur; it is even necessary to retain soundness.

Thus, when constructing Sk-minimal branches MINITAB does not initiate backtracking. A branch is considered complete and open if no more expansions can be performed or the γ-limit is reached. In this way a "complete" open tableau can be constructed. The comparison wrt. Sk-minimality of branches is performed during this construction and non-minimal branches are deleted at the earliest possible stage.

The second phase is a standard refutation step and could be performed by any first-order theorem prover with equality. Figure 8 gives a schematic over view of the two phases of MINITAB.

6 Conclusion

We have investigated two approaches to the general idea of domain minimization, namely domain circumscription with its intuitive semantics of minimal entailment and the tableau-based mini-consequence. We showed that mini-consequence lacks a model-theoretic semantics proper, but that a slight strengthening of mini-consequence, which we called Sk-minimal consequence, is sound and complete wrt. a stronger form of minimal entailment called variable minimal entailment. Thus, we were able to relate both approaches

on a formal level: In their strong forms domain circumscription and mini-consequence are equivalent.

We did not spend very much time on motivating the reasoning mode of domain minimization itself on more intuitive grounds. The reason is that this has been extensively done elsewhere, especially in the literature on circumscription, e.g. [McC80; EM87]. Nevertheless we showed that minimal entailment is too weak and that a stronger entailment notion is necessary. In an application of domain minimization to nonmonotonic temporal reasoning [Lor91] it turned out that variable minimal entailment is a promising candidate.

On the proof-theoretic side the advantage of Sk-minimal consequence over domain circumscription is its tableau-based proof theory which lends itself readily to an implementation. We discussed one such implementation called MINITAB, an order-sorted, free-variable tableau prover for variable minimal entailment.

Tableau systems have often been criticized for their relative inefficiency compared to, say, resolution-based provers. Much of this criticism does in fact apply to MINITAB because it has not been developed as a high speed theorem prover as such and hardly uses any optimization techniques. To investigate the applicability of optimization techniques that have been discussed in the literature (see e.g. [MR91], [BH92]) to tableau-based model construction is one of our objectives for further research.

References

[BH92] Bernhard Beckert and Reiner Hähnle. An improved method for adding equality to free variable semantic tableau. In D. Kapur, editor, *Proceedings of CADE-11.* Springer Verlag, 1992.

[EM87] David W. Etherington and Robert. E. Mercer. Domain circumscription: A reevaluation. *Computational Intelligence*, (3):94–99, 1987.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Springer Verlag, New York, 1990.

[Gin89] M. L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, (39):209–230, 1989.

[Hin88] Jaakko Hintikka. Model minimization - an alternative to circumscription. *Journal of Automated Reasoning*, (4):1–13, 1988.

[HS92] Reiner Hähnle and Peter Schmitt. The liberalized delta-rule in free variable semantic tableaux, 1992. submitted.

[Lor91] Sven Lorenz. Nonmonotonic temporal reasoning: Persistence, justified causation, and event minimization. In Z.W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems VI.* Springer Verlag, 1991.

[McC80] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, (13), 1980.

[McC86] John McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, (13):89–116, 1986.

[MR91] Neil V. Murray and Eric Rosenthal. On the relative merits of path dissolution and the method of analytic tableaux. Technical Report 90-5, SUNY at Albany, Albany, 1991.

[Prz89] T. C. Przymusinski. An algorithm to compute circumscription. *Artificial Intelligence*, (38):49–73, 1989.

[Sch87] P.H. Schmitt. The thot theorem prover. Technical Report 87.9.7, IBM Deutschland GmbH, Heidelberg, 1987.

Pierangelo Miglioli, Ugo Moscato, Mario Ornaghi
Information Science Department
University of Milan (Italy)

An improved refutation system for intuitionistic predicate logic

1. Introduction.

In this paper we present a refutation system for intuitionistic predicate logic (I.Pr.L.), which improves Fitting's one [1]. More precisely, in our system we introduce a new sign F_c beside the two signs T and F of [1]; the introduction of this new sign not only provides a more adequate basis to capture into the calculus the semantics of Kripke models for intuitionistic logic, but also makes considerably more efficient the calculus itself: it lowers the non determinism involved in the proofs since it explicitly states which rules may require "duplications of formulas" (i.e., which formulas "already used" in a step of a proof may be necessary also in subsequent steps).

The calculus was presented at the 6th Colloquium on Trees in Algebra and Programming [3]; here, we complete the technical results giving the proof of the completeness theorem for I.Pr.L. and show how the presence of the sign F_c allows to give an easy proof of Kolmogorov's theorem [4], in line with the fact that F_c "captures" into the system the features of intuitionistic negation. Another application of our idea will be a complete and correct calculus for the logic obtained by adding to I.Pr.L. Kuroda principle $\forall x\neg\neg A(x)\supset\neg\neg\forall xA(x)$ (see, e.g., [2]).

The treatment of all technical aspects is self contained; we only assume familiarity of the reader with Kripke models.

2. The calculus.

As we pointed out, our calculus works on "signed formula(s)" (s.f.) of the kind TA, FA, F_cA, with A any formula.

In order to make intuitive the meaning of the rules of the calculus, we give the following explanation of T, F, F_c in terms of their semantics with Kripke models [1,5,6]; here we adopt the formalism of [l], where, in a Kripke model $\underline{K} = \langle G,R,\models,P\rangle$, G is the set of nodes of $\underline{K}$, R is the reflexive and transitive accessibility relation on G (without loss of generality, one can assume that R is a partial ordering), $\models$ is the forcing relation and P is a function associating with every $\Gamma \in G$ a non empty domain (we call parameters the elements of such a domain).

Let $S = TA_1,\ldots,TA_n, FB_1,\ldots FB_m, F_cC_1,\ldots,F_cC_k$ be a set of s.f.;
we say that S is realized in $\Gamma \in G$ of $\underline{K}$ iff:

a) for every h, $1 \le h \le n$, $\Gamma \models A_h$

b) for every i, $1 \le i \le m$, $\Gamma \not\models B_i$

c) for every j, $1 \le j \le k$, $\Gamma \models \neg C_j$

The sign F_c (which stands for "certain falsehood") is stronger than the sign F (which stands for falsehood), and the presence of the two signs for falsehood makes explicit the presence of two semantical situations: global falsehood ($\Gamma \models \neg D$) and local falsehood ($\Gamma \not\models D$) respectively.

In the following, we will call "certain signed formula" any T or F_c signed formula, since it corresponds to stable forcing situations.

- A set S of s.f. is realizable iff there is some $\Gamma \in G$ of a model $\underline{K}$ s.t. Γ realizes S.

- By a configuration we mean a sequence $S_1/S_2/\ldots/S_j/\ldots$, where every S_j is a set of s.f.; a configuration is realizable iff at least a S_j is realizable.

Now we explain the calculus; the rules are given in Table 1.

Table 1

T-RULES	**F-RULES**	**F_c-RULES**
$\dfrac{S,T(A\wedge B)}{S,TA,TB}\ T\wedge$	$\dfrac{S,F(A\wedge B)}{S,FA/S,FB}\ F\wedge$	$\dfrac{S,F_c(A\wedge B)}{S_c,F_cA/S_c,F_cB}\ F_c\wedge$
$\dfrac{S,T(A\vee B)}{S,TA/S,TB}\ T\vee$	$\dfrac{S,F(A\vee B)}{S,FA,FB}\ F\vee$	$\dfrac{S,F_c(A\vee B)}{S,F_cA,F_cB}\ F_c\vee$
$\dfrac{S,T(A\supset B)}{S,FA,T(A\supset B)/S,TB}\ T\supset$	$\dfrac{S,F(A\supset B)}{S_c,TA,FB}\ F\supset$	$\dfrac{S,F_c(A\supset B)}{S_c,TA,F_cB}\ F_c\supset$
$\dfrac{S,T(\neg A)}{S,F_cA}\ T\neg$	$\dfrac{S,F(\neg A)}{S_c,TA}\ F\neg$	$\dfrac{S,F_c(\neg A)}{S_c,TA}\ F_c\neg$
$\dfrac{S,T\forall xA(x)}{S,TA(a),T\forall xA(x)}\ T\forall$	$\dfrac{S,F\forall xA(x)}{S_c,FA(a)}\ F\forall$ *with* a *new*	$\dfrac{S,F_c\forall xA(x)}{S_c,FA(a),F_c\forall xA(x)}\ F_c\forall$ *with* a *new*
$\dfrac{S,T\exists xA(x)}{S,TA(a)}\ T\exists$ *with* a *new*	$\dfrac{S,F\exists xA(x)}{S,FA(a)}\ F\exists$	$\dfrac{S,F_c\exists xA(x)}{S,F_cA(a),F_c\exists xA(x)}\ F_c\exists$

Explanation and definitions.

- Every rule of Table 1 is applied to a formula of a set S_i occurring in a configuration $S_1/\ldots/S_i/\ldots$; e.g., the notation S, TA∧B points out that the rule T∧ is applied to the formula TA∧B of the set S∪{TA∧B}, where S is possibly empty.

- The rules F⊃, F¬, F∀, $F_c\wedge$, $F_c\supset$, $F_c\neg$, $F_c\forall$ narrow S to S_c , where S_c is obtained from S by deleting all the F s.f. (in other words, S_c is the certain part of S).

- The rules T⊃, T∀, $F_c\forall$, $F_c\exists$ may give rise to duplications (i.e., the s.f. to which the rule is applied is copied under the line); all the other rules of Table 1 can't give rise to any duplication.

- A proof-table is a sequence of applications of the rules of Table 1 starting from some configuration; a proof-table is closed iff all the sets S_i of the final configuration are contradictory, i.e. they contain TA and FA or TA and F_cA for some A of S_i . A proof of a formula D is a closed proof-table starting with F D.

- A finite set S of s.f. is consistent iff no proof-table starting with S is closed; if S is infinite, it is consistent iff every finite subset S' of S is consistent.

As one can see in a detailed proof of the validity theorem, the reason why one has to narrow S to S_c in some rules is related to the meaning of the logical constants $\supset$, $\neg$ and $\forall$ in terms of Kripke models; roughly speaking, the formula (formulas) obtained by the application of one of these rules is (are) realized in some node Δ connected with the node Γ realizing the set of s.f. before the application of the rule; therefore, Δ realizes only the certain part of S.

A refutation calculus for classical predicate logic is obtained by modifying Table 1 as follows:

1) we have only T and F_c s.f. (so, $S=S_c$ for any S);

2) the F-rules are deleted;

3) the rules T $\supset$ and $F_c\forall$, which introduce F s.f. and impose duplications, are so modified: no duplication is required and the F s.f. become F_c s.f.;

4) a proof of a formula A is a closed proof-table starting with F_cA.

Remark that we stated to characterize the classical calculus by means of T and F_c rules since in the classical case (i.e. in the class of Kripke models whose G contain only one element) one cannot have "uncertain forcing situation". A complete calculus could also be given using T and F rules, imposing a duplication in the F$\exists$ rule and preserving S after the application of any rule.

Now, we give an example where the use of the sign F_c avoids duplications which are necessary in Fitting's system; let's prove the intuitionistic theorem $\neg\neg(A\supset((B\vee\neg B)\wedge A))$:

$F\neg\neg(A\supset((B\vee\neg B)\wedge A))$

$T\neg(A\supset((B\vee\neg B)\wedge A))$

$F_c(A\supset((B\vee\neg B)\wedge A))$

$TA,F_c((B\vee\neg B)\wedge A)$

$TA,F_cB\vee\neg B/TA,F_cA$

$TA,F_cB,F_c\neg B/TA,F_cA$

$TA,F_cB,TB/TA,F_cA$

If we prove the same theorem using Fitting's calculus, the proof-table becomes:

$$F\neg\neg(A\supset((B\vee\neg B)\wedge A))$$

$$T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$F(A\supset((B\vee\neg B)\wedge A)),T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,F((B\vee\neg B)\wedge A),T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,FB\vee\neg B,T\neg(A\supset((B\vee\neg B)\wedge A))/TA,FA,T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,FB,F\neg B,T\neg(A\supset((B\vee\neg B)\wedge A))/TA,FA,T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,TB,T\neg(A\supset((B\vee\neg B)\wedge A))/TA,FA,T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,TB,F(A\supset((B\vee\neg B)\wedge A))/TA,FA,T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,TB,TA,F((B\vee\neg B)\wedge A)/TA,FA,T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,TB,FB\vee\neg B/TA,TB,FA/TA,FA,T\neg(A\supset((B\vee\neg B)\wedge A))$$

$$TA,TB,FB,F\neg B/TA,TB,FA/TA,FA,T\neg(A\supset((B\vee\neg B)\wedge A))$$

We conclude this paragraph with the example of an unavoidable duplication; the proof-table of the intuitionistic theorem $((A\vee\neg A)\supset B)\supset\neg\neg B$ is:

$$F((A\vee\neg A)\supset B)\supset\neg\neg B$$

$$T(A\vee\neg A)\supset B,F\neg\neg B$$

$$T(A\vee\neg A)\supset B,T\neg B$$

$$T(A\vee\neg A)\supset B,F_cB$$

$$F_cB,FA\vee\neg A,T(A\vee\neg A)\supset B/F_cB,TB$$

$$F_cB,FA,F\neg A,T(A\vee\neg A)\supset B/F_cB,TB$$

$$F_cB,TA,T(A\vee\neg A)\supset B/F_cB,TB$$

$$F_cB,FA\vee\neg A,TA/F_cB,TA,TB/F_cB,TB$$

$$F_cB,FA,F\neg A,TA/F_cB,TB,TA/F_cB,TB$$

This shows that for the rule $T\supset$ duplications are, in general, unavoidable. Examples of unavoidable duplications can be provided also for the rules $T\forall$, $F_c\forall$, $F_c\exists$; thus (in view of the completeness theorem below) our calculus requires only the necessary duplications.

3. Validity and completeness of I.Pr.L.

The refutation system given in Table 1 is correct and the validity theorem has the following form:

If a proof-table of a s.f. FA is closed, then A is valid.

The proof of this theorem is based on the fact that the rules of the calculus preserve realizability; i.e., if a configuration is realized in a node of a Kripke model $\underline{K}$, then the configuration obtained by applying to the former configuration a rule is realized in a node of $\underline{K}$; so, if a proof-table of FA is closed, then FA isn't realizable, and A is forced in all the nodes G of all the Kripke models $\underline{K}$. We omit the proof, which doesn't present particular difficulties.
The completeness theorem has the following form:

If a formula A is valid, then there is a closed proof-table of FA.

The proof is based on the following lemma:

Lemma 1. If a set I of s.f. is consistent, then there is a $\underline{K}$ which realizes I.

The proof of Lemma 1 is based on a technique which allows to construct a Kripke model $\underline{K}_I$ starting from a consistent set of s.f. I. Before explaining the construction of $\underline{K}_I$ we give some preliminary definitions:

- A reference set of a set of s.f. S is a non empty set of parameters containing all the parameters occurring in S.

- Given a consistent set of s.f. S containing a s.f. H, we call a consistent reduced of H in S a set R containing one or two formulas obtained by applying the rule related to H, with the following convention: if the rule is a "splitting" rule, i.e. it gives rise to a configuration of two sets, then we choose a consistent one (at least one exists) in such a way that $\{S - H\} \cup R$ is consistent.

Now we come to the construction of $\underline{K}_I$; this will be done in two main steps A and B. Step A constructs two consistent sets S* and $\overline{S}$ starting from a consistent set of s.f. S; S* is called a fulfilled set of S and $\overline{S}$ is called a related set of S; $\overline{S}$ will be used to construct a node Γ of $\underline{K}_I$. Step B constructs sets S' named associated of S; the related $\overline{S}'$ of an associated S' will be used to construct a Δ of $\underline{K}_I$ immediately connected with Γ by the accessibility relation of $\underline{K}_I$. The definitions of steps A and B are strictly related to the rules of the calculus.

Step A.

Let:

$l_1 = A_1, \ldots, A_n, \ldots$ be any listing of the s.f. of S (without duplications of formulas);

$l_2 = p_1, \ldots p_m, \ldots$ be any listing of the parameters of a reference set of S.

Starting from l_1 and l_2 , we construct the following collections S_i of s.f. and π_i of parameters:

S_0 and π_0 are empty;

$S_{i+1} = \cup_{H_j \in S_i} R(H_j,i) \cup \{A_{i+1}\}$ (being $S_i = \{H_1, \ldots, H_k\}$)

$\pi_{i+1} = \pi_i \cup \{p_{i+1}\} \cup new(S_i, A_{i+1})$

where $R(H_j,i)$ is so defined:

1) if H is of the kind $TA\wedge B$, $TA\vee B$, $T\neg A$, $TA\supset B$, $FA\wedge B$, $FA\vee B$, $F_cA\vee B$, then $R(H_j,i)$ is a consistent reduced of $H_j\in S_i=\{H_1,\ldots,H_k\}$ in $R(H_1,i)\cup\ldots\cup R(H_{j-1},i)\cup\{H_j,\ldots,H_k,A_{i+1},A_{i+2}\ldots\}$;

2) if H is $T\forall xA(x)$, then $R(H_j,i)=\{TA(q_1),\ldots,TA(q_r),T\forall xA(x)\}$, where $\{q_1,\ldots,q_r\}=\pi_i$;

3) if H is $F_c\exists xA(x)$, then $R(H_j,i)=\{F_cA(q_1),\ldots,F_cA(q_r),F_c\exists xA(x)\}$;

4) if H is $T\exists xA(x)$, then $R(H_j,i)=\{TA(q)\}$, where q is a <u>new</u> parameter; here "new q" means that $q\notin\pi_i$, $q\notin l_2$ (in particular one has that $q\neq q'$ for any two q,q' introduced by 4));

5) in all the other cases, $R(H_j,i)=\{H_j\}$;

and new (S_i, A_{i+1}) is the set of the new parameters introduced by 4).

Now, we can define the fulfilled set S*, the related set $\overline{S}$ and π^*:

<u>Def.1.</u>

- $S^*=\cup_i S_i$
- $\overline{S}$ = {H/H$\in$ S* and H is final} where we call final a formula H of the kind $T\forall xA(x)$, $F_c\exists xA(x)$, $TA\supset B$ (i.e.the formulas of the above points 1), 2), 3) which duplicate) and the formulas of the above point 5)
- $\pi^*=\cup_i\pi_i$

One has:

a) $\overline{S}\subseteq S^*$;

b) $\overline{S}$ is consistent (since if H is a final s.f. of $\overline{S}$, then there is some k such that $H\in S_i$ for every $i\geq k$);

c) π^* is a reference set of $\overline{S}$

Step B.

The associated sets are defined as follows:

<u>Def.2.</u> Let $\overline{S}$ be a related set of a consistent set of s.f. S and let A be a s.f. belonging to $\overline{S}$ of the kind $F\neg B$, $F_c\neg B$, $F_cB\supset C$, $FB\supset C$, $F_cB\wedge C$, $F\forall xB(x)$, $F_c\forall xB(x)$; the set S' obtained from $\overline{S}$ by substituting A with one of its consistent reduced in $\overline{S}$ and deleting all the F s.f. of $\overline{S}$ is said to be a simple associated of $\overline{S}$.

<u>Def .3.</u> Let $\overline{S}$ be the related of a consistent set of signed formulas S and let A be a formula of $\overline{S}$ of the kind $F\exists xB(x)$; then, for every parameter $a\in\pi^*$, the set S' obtained from $\overline{S}$ by substituting in $\overline{S}$ the formula $F\exists xB(x)$ with the formula FB(a) is said to be a F$\exists$-associated of $\overline{S}$.

Henceforth, with the term "associated set" we will indicate "simple associated" and "F$\exists$-associated" set.

Now, the construction of a Kripke model $\underline{K_I}=<G_I, R_I, \models_I, P_I>$ starting from a consistent set of s.f. I can be given as in Def. 4 below. We stipulate the following conventions:

- the elements of G_I will correspond to suitable related sets $\overline{S}$ of s.f.; we will identify the elements of G_I with the corresponding $\overline{S}$;

- for every $\overline{S}$, the set of its parameters (individuals) $P_I(\overline{S})$ will be the set π^* obtained in Step A to construct S* ; furthermore, in the construction of an associated S' of a $\overline{S}$ we will start with a reference set containing the parameters of $\overline{S}$ in such a way that π'^* will contain π^*.

Def.4. Given a consistent set of s.f. I, we define a Kripke model $\underline{K}_I$ as follows:

I) $\underline{K}_I$ is a tree (possibly infinite) whose root is a related $\overline{I}$ of I;

II) having defined a node $\overline{S}$ of $\underline{K}_I$, consider a generic associated S' of $\overline{S}$ and let $\overline{S}'$ be a related of S'; then $\overline{S}'$ will be a node of $\underline{K}_I$ immediately accessible from $\overline{S}$ (there are as many nodes of this kind as the associated of $\overline{S}$ are; the parameters of $\overline{S}$ ' are obtained according to the above conventions);

III) for every node $\overline{S}$ of $\underline{K}_I$, we force in it all the atomic T s.f. belonging to $\overline{S}$. The forcing relation so defined is well given, i.e., if $\overline{S}'$ follows $\overline{S}$, then the set of its atomic T s.f. contains the corresponding set of $\overline{S}$ and the same holds for the parameters. The forcing relation is extended in the usual way to any formula.

Remark that the accessibility relation, as it has been defined, is such that, given two nodes $\overline{S}_1$ and $\overline{S}_2$ of $\underline{K}_I$ s.t. $\overline{S}_1$ follows $\overline{S}_2$, the path of $\underline{K}_I$ connecting $\overline{S}_1$ with $\overline{S}_2$ is finite.

According to Def.3 the rule F$\exists$ gives rise to new states of $\underline{K}_I$ (as the rules considered in Def.2 and differently from the rules considered in Step A). This treatment of F$\exists$ is crucial in order to prove that it doesn't require duplications. Indeed, one could give a correct and complete F$\exists$ rule with duplications for I.Pr.L.; in this case, such a rule could be treated in Step A.

Now the lemma has the following form:

Lemma 1* . For every consistent set of s.f. I, for every Kripke model $\underline{K}_I$ constructed as Def.4 prescribes, for every node $\overline{S}$ of $\underline{K}_I$, and for every formula A belonging to the fulfilled S* of S (being S the set whose related is $\overline{S}$), A is realized in $\overline{S}$.

The proof is by induction on the complexity of the formulas; in the proof we use s as one of the signs T, F, F_c.

Basis. Let $sA \in S^*$ of complexity zero; then, if s=T, TA is realized in $\overline{S}$ by our definition of forcing; if s=F, then, since $\overline{S}$ is consistent, TA cannot belong to $\overline{S}$; hence FA is realized in $\overline{S}$; if $s=F_c$, F_cA belongs to $\overline{S}$ and hence to every $\overline{S}_1$ subsequent to $\overline{S}$; so, $\overline{S}$ realizes F_cA.

Step. Suppose, by inductive hypothesis, that the lemma holds for formulas of complexity less than n. We prove that it holds also for formulas of complexity n.

It is convenient to divide the proof of the step by grouping the formulas in five sets, the first one containing formulas of the kind $T\neg B$, $TB\vee C$, $TB\wedge C$, $FB\wedge C$, $FB\vee C$,

$F_cB \vee C$, $T\exists xB(x)$, the second $F\neg B$, $FB \supset C$, $F\exists xB(x)$, $F\forall xB(x)$, the third $F_cB \wedge C$, $F_c\neg B$, $F_cB \supset C$, the fourth $F_c\forall xB(x)$, $F_c\exists xB(x)$, $T\forall xB(x)$ and the fifth $TB \supset C$. We have chosen this partition because, for every formula of the first, second and third group the proof of the lemma can be carried out in a similar way.

First Group. The scheme of the proof for a formula H of this group looks as follows: $H \in S^*$ implies $R(H,i) \subseteq S^*$ (for some i) and, by the inductive hypothesis on (the destructured formulas of) R(H,i) one immediately obtains that $\overline{S}$ realizes H.

Second Group. The scheme of the proof for a formula H of this group looks as follows: $H \in S^*$ implies $H \in \overline{S}$ which implies that a reduced set R of H in $\overline{S}$ is contained, by Step B, in $S' \subseteq S'^*$ and, by the inductive hypothesis, $\overline{S}'$ realizes R and so, immediately, $\overline{S}$ realizes H.

Third Group. The scheme of the proof for a formula $H = F_cD$ of this group looks as follows: to prove that $\overline{S}$ realizes F_cD is to prove that, for every $\overline{S}_1$ s.t. $\overline{S} R_I \overline{S}_1$, $\overline{S}_1 \not\models_I D$. We have two cases:

a) $F_cD \in \overline{S}_1$; then, $\exists\, \overline{S}_2$ s.t. $\overline{S}_1 R_I \overline{S}_2$ and $\overline{S}_2$ (and hence $\overline{S}_2^*$) contains a consistent reduced R of F_cD in $\overline{S}_1$; by the inductive hypothesis, one can immediately prove that $\overline{S}_2 \not\models_I D$ and so $\overline{S}_1 \not\models_I D$;

b) $F_cD \notin \overline{S}_1$; then $\exists\, \overline{S}_2$ and $\overline{S}_3$ s.t. $\overline{S} R_I \overline{S}_2 R_I \overline{S}_3 R_I \overline{S}_1$ s.t.
$F_cD \in \overline{S}_2$ and $\overline{S}_3$ contains a consistent reduced R of F_cD in $\overline{S}_2$: by the inductive hypothesis one can immediately prove that $\overline{S}_3 \models_I \neg D$ and so $\overline{S}_1 \not\models_I D$.

Fourth Group. For a formula H of the kind $T\forall xB(x)$ or $F_c\exists xB(x)$ one can apply the following scheme: $H \in \overline{S}$ implies (as an effect of duplication) that, for every $\overline{S}_1$ s.t. $\overline{S} R_I \overline{S}_1$, $H \in \overline{S}_1$ and, for every $p \in \pi_1^*$, there exists i s.t. R(H,i) contains TB(p) (in the case of $T\forall xB(x)$) or $F_cB(p)$ (in the case of $F_c\exists xB(x)$); by the inductive hypothesis one immediately obtains that $\overline{S}_1$ realizes H.

For $H = F_c\forall xB(x)$, $H \in \overline{S}$ implies that, for every $\overline{S}_1$ s.t. $\overline{S} R_I \overline{S}_1$, $H \in \overline{S}_1$ (by duplication) and there exist $\overline{S}_2$ s.t. $\overline{S}_1 R_I \overline{S}_2$ and a $p \in \pi_2^*$ s.t. $FB(p) \in \overline{S}_2^*$; so, by the inductive hypothesis, FB(p) is realized, and the result immediately follows.

Fifth Group. To prove that $\overline{S}$ realizes $TB \supset C$ is to prove that, for every $\overline{S}_1$ s.t. $\overline{S} R_I \overline{S}_1$, $\overline{S}_1 \not\models_I B$ or $\overline{S}_1 \models_I C$; if the involved reduced set $R(TB \supset C,i)$ contains TC (so that S^* contains TC) the lemma immediately follows by the inductive hypothesis. Otherwise $R(TB \supset C,i)$ contains FB and $TB \supset C$, and one applies a scheme similar to the one of the rules of Third Group (taking into account $TB \supset C$ instead of F_cD) distinguishing the case where $TB \supset C \in \overline{S}_1$ (in which case $FB \in S_1^*$) from the case where $TB \supset C \notin \overline{S}_1$ (in which case $TC \in S_1^*$).

Now we are able to prove the

Completeness Theorem for I.Pr.L.

If a formula A is valid in I.Pr.L., then it is a theorem of I.Pr.L.

Suppose that A is not a theorem; then, {FA} is a consistent set of s.f.; but, by the lemma just proved, any consistent set of s.f. is realizable, hence {FA} is realizable, which is absurdum.

Let us see now some applications of the calculus which put into evidence the role of the sign F_c.

We begin with the following two facts:

*) One can easily see that if the rule $T\supset$ is substituted by the following weaker rule:

$$\frac{S,TA\supset B}{S,F_cA/S_c,TB}\ \overline{T\supset}$$

the calculus is sound (but not complete) for I.Pr.L.

The following proposition is an immediate consequence of the above point *) (remark that if one has only certain s.f., then $\overline{T\supset}$ coincides with the classical $T\supset$):

Proposition 1. In the propositional case, any classical proof-table is a proof-table of the intuitionistically sound (but not complete) calculus containing $\overline{T\supset}$ instead of $T\supset$.

From Proposition 1 the well known Kolmogorov's theorem [4] follows:

$\vdash_{C.L.} A \Leftrightarrow \vdash_{I.L.} \neg\neg A$ with A a propositional formula (being $\vdash_{C.L.}$ the provability in classical propositional logic and $\vdash_{I.L.}$ the provability in intuitionistic propositional logic).

For, an intuitionistic proof-table of the formula $\neg\neg A$ will start with $F\neg\neg A$ and its second configuration will contain $T\neg A$; so, the third configuration will contain F_cA and for the subsequent configurations Proposition 1 applies.

As it is well known, this theorem cannot be extended to I.Pr.L.; in our calculus, the counter-part of this fact corresponds to the following point **):

**) If one substitutes the rule $F_c\forall$ with the following:

$$\frac{S,F_c\forall A(x)}{S_c,F_cA(a)}\ \overline{F_c\forall}\ \text{with a new}$$

the calculus is no longer sound.

The predicative extension of Kolmogorov's theorem is given by the following result for the well known "negative translation" [7]:

for any A, $\vdash_{C.Pr.L.} A \Leftrightarrow \vdash_{I.Pr.L.} A'$ being A' the negative translation of A and C.Pr.L the classical predicate logic.

Also this result can be achieved in our attitude by remarking that:

i) the translation A' of any A is an Harrop formula satisfying the following properties:

1) every atomic subformula of A is in the scope of a negation;

2) if A' is different from $\neg\neg$ p, with p atomic, the main subformulas of A' are translations of some formulas;

ii) the rule $\overline{F_c\ \forall}$ is intuitionistically sound for any Harrop formula A' satisfying the above Point i); this follows from the fact that, for any $\Gamma \in G$ of a Kripke model $\underline{K}$, if $\Gamma \not\models A'$, then there is a $\Delta \in G$ s.t. $\Gamma R \Delta$, $\Delta \models \neg A'$ (the proof is by induction on the complexity of A');

iii) if (for any A) $\vdash_{C.Pr.L.}$ A then $\vdash_{C.Pr.L.}$ A', where A' is the negative translation of A.

From i), ii) and iii) the analogous of Proposition 1 can be proved for I.Pr.L. taking into account the negative translation A'.

The above short examples show how the use of the new sign F_c (together with its meaning in terms of Kripke models) allows to naturally obtain in a simple way some well known results about intuitionistic logic. We think also that our approach can be profitably applied to other contexts (e.g. modal logics, intermediate logics, and so forth), where different forcing situations may be captured by different signs and by modifying the set (sets) of formulas produced by the applications of the rules.

As a further example, if one considers the intermediate logic **K** obtained by adding to I.Pr.L. the well known Kuroda principle [2]

(K) $\forall x \neg\neg A(x) \supset \neg\neg \forall x A(x)$.

One has that $\vdash_K \neg\neg A$ iff $\vdash_{CPr.L.}$ A

The Kripke models for this logic are the ones of I.Pr.L. with the following additional property: for every node G there is a G' s.t. G R G' and G' is "terminal" (i.e., G' R G" implies G'=G"). Since terminal nodes behave as "classical" nodes, it turns out that the rule $\overline{F_c\ \forall}$ is sound for **K**. With a proof quite similar to the one of the above Completeness Theorem for the intuitionistic calculus, we can prove that the calculus obtained by substituting F_c" with $\overline{F_c\ \forall}$ and leaving unchanged the other rules is complete for **K**. Notice that the rule $\overline{F_c\ \forall}$ doesn't duplicate. Hence the calculus for **K** has a lower degree of non determinism than the intuitionistic one, and to prove in **K** becomes easier.

REFERENCES

[1] M.C. Fitting, Intuitionistic logic, model theory and forcing, North-Holland, Amsterdam, 1969.

[2] D. Gabbay, Semantical investigations in Heyting's intuitionistic logic, Reidel, Dordrecht, 1981.

[3] P. Miglioli, U. Moscato and M. Ornaghi, Trees in Kripke models and in an intuitionistic refutation system, in E. Astesiano, C. Böhm (ed.), CAAP '81, Lecture Notes in Computer Science, Vol.112, Springer-Verlag, Berlin, 1981, pp. 316-330.

[4] A. Kolmogorov, O principe tertium non datur (On the principle of tertium non datur), Mat. Sb., 32, 1925, pp. 646-667 (Russian) [English translation in: J. van Heijenoort (ed.), From Frege to Gödel, Harvard University Press, Cambridge, MA, 1967, pp. 414-437].

[5] S. Kripke, Semantical analysis of intuitionistic logic I, in Crossley J. N. and Dummett M. A. E. (ed.), Formal systems and recursive functions, North-Holland, Amsterdam, 1965, pp. 92-129.

[6] C. A. Smorinsky, Applications of Kripke models, in A.S. Troelstra (ed.), Metamathematical investigation of intuitionistic arithmetic and analysis, Lecture Notes in Mathematics, Vol.344, Springer-Verlag, Berlin, 1973, pp. 324-391.

[7] A.S. Troelstra Aspects of constructive mathematics in J. Barwise (ed.), Handbook of mathematical logic, North Holland, Amsterdam, 1977, pp. 973-1052.

Kripke and Relational-Style Semantics and Associated Tableau Proof Systems for Arbitrary Finite Valued Logics

(Abstract)

Charles G. Morgan
Department of Philosophy
University of Victoria
Victoria,B.C. CANADA V8W 3P4
email: morgan@uvphys.phys.uvic.ca

Ewa Orlowska
Institute of Computer Science
Polish Academy of Sciences
P.O. Box 22 00-901 WARSAW PKiN
POLAND
email: orlowska@plearn.bitnet

I. Introduction

In previous work, Orlowska (1987, 1988, 1990, 1991) established a relationship between several nonclassical logics and relation algebras. In particular, it was demonstrated that formulas in various propositional logics can be treated as relations, with the propositional connectives being treated as relational operations. Further, the relational semantics were used as a basis for tableau proof systems which can be said to constitute a proof theory for relational logic. This development of relational logic allowed for the uniform treatment of the various nonclassical logics. In Morgan (1991), the implementation and elementary testing of an automated theorem prover for relational logic was reported and discussed.

To date, the logics analyzed into relation algebras include various modal logics, intuitionistic logic, dynamic logic, and Post logics. In this paper we present a uniform analysis of arbitrary finite valued logics into a classical relational system. We are particularly interested in formulations of the sort discussed in Rosser and Turquette, since (a) those formulations permit the expression of semantic values directly in the syntax, and (b) the logic of any collection of finite valued operators can be axiomatized as a conservative extension of one basic system. The treatment of many valued logics by the logic of relations permits a natural extension to many valued truth-functional logics to many valued indexical logics of the sort discussed in Morgan (1979).

II. The mv language and standard <n,s> mv semantics

We take our language to be based on a denumerable set VAR of propositional variables and a family $\{c^i_j\}$ (i and j positive, finite) of i-component propositional connectives. 0-component connectives are just propositional constants. The set FOR of formulas includes VAR and is closed under the propositional connectives.

For the standard many-valued semantics, we take the semantic range to be n values indexed by the integers SR = {0, 1, ..., s, ..., n-1}, for $0 < s \leq n-1$. The designated values are {s, ..., n-1}, and the others are said to be undesignated. (We are treating the low values as undesignated and the high values as designated, which is the opposite of Rosser and Turquette (1952). However, it seems more natural to think of the high values as "good", and this order parallels both probability theory and fuzzy logic.) With the family of connectives $\{c^i_j\}$, we associate a family of semantic functions $\{f^i_j\}$, each being a map from SR^i into SR. A "model" is just an interpretation function I that assigns to each propositional variable a value in the semantic range. Values are assigned to all expressions in the usual way.

The language is assumed to contain monadic operators J_i, for $0 \leq i \leq n-1$, one for each value in SR. Intuitively, $J_i(A)$ says that A takes value i. Semantically, the value of $J_i(A)$ is n-1 if the value of A is i, and 0 otherwise. The language is also assumed to contain a conditional operator $\supset$, which satisfies "standard conditions". That is, the semantics of the operator is such that if all designated values are mapped to "true" and all undesignated values are mapped to "false", the resulting table would be that of the classical material conditional.

III. Kripke-style <n,s> mv semantics

For the Kripke-style semantics, we first define a frame to be an ordered tuple $<W, SR, s, \{f^i_j\}>$, where W is a non-empty set of worlds, and as above, SR is the semantic range with critical value s, and $\{f^i_j\}$ is a family of i-place functions from SR^i into SR. A model is then defined to be a tuple $<W, SR, s, \{f^i_j\}, I, \{S_k, k \text{ in } SR\}>$ consisting of a frame plus an interpretation function I for variables and a set of meaning functions S_k, one for each k in SR. The function I is a map from VAR × W into SR. The functions S_k are maps from FOR × W into SR. Intuitively, $S_k(A)$ is the set of worlds at which A takes value k. The functions S_k are defined inductively as follows:

$$S_k(p) = \{w: I(p,w) = k\}, \text{ for p in VAR}$$
$$S_k(c^i_j(A_1,...,A_i)) = \bigcup\{S_{m1}(A_1) \cap ... \cap S_{mi}: f^i_j(m1,...mi) = k\}$$

The standard notions of being satisfied at a world in a model, being satisfied in a model, and of being valid are all defined in the usual way. Initially, we limit our attention to connectives that are defined locally at each world; many valued indexical connectives will generally require the introduction of one or more accessibility relations into our semantics.

IV. Relational logic and relational <n,s> semantics

The language of our relational logic includes the denumerable set CONREL of relational constants, a family $\{o^i_j\}$ of i-argument relational operators, including the relational operators - (complement), ∪ (union) and ∩ (intersection). The set of relational expressions EREL includes CONREL and is closed with respect to the relational operators. The language includes a set VAROB of individual (object) variables. The set relational formulas FORREL includes just strings of the form x A y for x and y in VAROB and A in EREL.

A model in the relational semantics is defined to be a tuple as follows: <OB, SR, s, $\{r_{k,p}\}$, $\{f^i_j\}$, $\{Z_k\}$>. Items SR, s, and f^i_j are as above. OB is a non-empty set of objects. For p in CONREL and k in SR, each $r_{k,p}$ is an ideal on OB; that is, $r_{k,p}$ is of the form O × OB, for O a subset of OB. For k in SR, Z_k is a function from EREL into the powerset of OB × OB, and is defined inductively on relational expressions as follows:

$Z_k(p) = r_{k,p}$
$Z_k(- A) = - Z_k(A)$
$Z_k(A \cup B) = Z_k(A) \cup Z_k(B)$
$Z_k(A \cap B) = Z_k(A) \cap Z_k(B)$
$Z_k(o^i_j(A_1,...,A_i)) = \bigcup\{Z_{m1}(A_1) \cap ... \cap Z_{mi}(A_i): f^i_j(m1,...,mi) = k\}$

As usual, a valuation is just a mapping v from VAROB into OB. A valuation on a model satisfies the relational formula x A y if and only if <v(x),v(y)> is in $\bigcup\{Z_k(A)$: k designated$\}$. A relational formula is valid on a model just in case it is satisfied by all valuations on the model.

V. Relational proof systems

We are particularly interested in languages of the sort discussed in Rosser and Turquette (1952). Thus we assume the presence in the language of the monadic auto-descriptivity operators J_k. The tableau rules for union, intersection, and complement are the standard ones. With the use of the J-operators, it is possible to specify tableau rules for arbitrary relational operators o^i_j.

Intuitively, the analysis above treats each expression A of the logic as corresponding to a vector of sets $<S_0(A), ..., S_{n-1}(A)>$, one set for each of the values in SR; set $S_k(A)$ would be the set of worlds at which A takes value k. The S_k obviously correspond to the monadic J_k operators. As an alternative, we could treat an expression A as corresponding to a vector $<D_0(A), ..., D_{n-1}(A)>$ where $D_i(A) = \{w: V(A,w) \geq i\}$. The D operators are standard in Post logics, and this analysis would allow us to treat arbitrary finite valued logics by means of the proof systems for Post relation algebras as in Orlowska (1991). As another alternative, we

note that Rosser and Turquette (1952) give uniform axiomatizations of arbitrary finite valued logics by conservatively embedding them in a language including the J operators and one conditional operator satisfying standard conditions. The axioms for the conditional operator are quite weak, and correspond to the implicational fragment of intuitionistic logic. This logic can be treated semantically by a well known Kripke style semantics. Following this avenue, we would again come up with a system similar to that for the Post relational algebras. The conditional and the J operators of the Rosser and Turquette formulation may all be treated as "regular" connectives in the sense of Hähnle (1991). Thus it may be possible to simplify the tableau treatment of many valued indexical operators in the way suggested there.

References

Hähnle, Reiner. (1991) Uniform notation of tableau rules for multiple-valued logics. Proceedings of the Twenty-First International Symposium on Multiple-Valued Logic, IEEE press, pp. 238-245.

Morgan, Charles. (1992) An automated theorem prover for relational logic. Presented at Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Lautenbach. Forthcoming.

Morgan, Charles. (1979) Local and global operators in many-valued modal logics. Notre Dame Journal of Formal Logic, 20, pp. 401-411.

Orlowska, Ewa. (1987) Relational interpretation of modal logics. To appear in: H. Andreka, D. Monk, and I. Nemeti. Algebraic Logic. North Holland

Orlowska, Ewa. (1988) Dynamic logic with program specifications and its relational proof system. Bulletin of the Section of Logic, 18, 132-137.

Orlowska, Ewa. (1990) Interpretation of relevant logics in a logic of ternary relations. Bulletin of the Section of Logic, 19, 39-48.

Orlowska, Ewa. (1991) Post relation algebras and their proof system. Forthcoming.

Rosser, J. Barkley, and Turquette, Atwell R. (1952) Many-Valued Logics. North-Holland Publishing Company.

On The Relative Merits of Path Dissolution and the Method of Analytic Tableaux†

Neil V. Murray
Inst. for Programming & Logics
Dept. of Computer Science
State Univ. of N.Y. at Albany
Albany, NY 12222
nvm@cs.albany.edu

Erik Rosenthal
Dept. of Mathematics
University of New Haven
300 Orange Avenue
West Haven, CT 06516
brodsky%nhu.UUCP@yale.edu

ABSTRACT

Path dissolution is an inferencing mechanism that generalizes the method of analytic tableaux. We present several results demonstrating that tableau deductions can be substantially speeded up with applications of dissolution technology. We also consider the class of formulas on which the method of analytic tableaux was first shown to be intractable and prove that, with the application of the ordinary distributive law, standard tableau methods admit linear time proofs for this class.

1. Introduction

Tableau methods were originally developed and studied by a number of logicians, among them Beth [1], Hintikka [12], and Smullyan [21], who built on the work of Gentzen [9]. It is probably Smullyan who is most responsible for popularizing these methods; his particularly elegant variation on these techniques is known as the method of *analytic tableaux*. More recently, tableau methods have been receiving considerable attention from researchers investigating both automated deduction and logics for artificial intelligence; this includes both implementors and those whose focus is primarily theoretical [3,5,7,8,10,19,20,21,22].

Path dissolution, introduced in [14], is a generalization of these methods. In this paper, we examine the tableau paradigm and present several results demonstrating that tableau deductions can be substantially speeded up with applications of dissolution technology. We also consider the class of formulas on which the method of analytic tableaux was shown to be intractable by Cook and Reckhow ([4]). It turns out that, with appropriate use of factoring (i.e., the distributive law), standard tableau methods can handle this class in linear time. This is of interest since employing the standard distributive laws is not a fundamentally greater enhancement to deduction methods that do not rely on clause form than is the use of associativity, commutativity, and merging in methods, such as resolution, that do.

The reader should be forewarned that there is a potentially misleading difference in emphasis between the typical descriptions of the tableau method and the one presented here. Tableau deductions are usually cast as tree structures in which paths may grow through the addition of new *lines* in the tree; the number of paths may increase due to *splitting* or *branching* operations. The lines and branch points of a tableau proof tree are meta-linguistic representations of conjunction and disjunction, respectively. In this paper, we strip them of their special status; a tableau proof tree then becomes merely a single formula.

In the next section we provide a brief review of path dissolution; for more detail see [14]. We also cast analytic tableau methods in the framework of semantic graphs. Section 3 begins with an example that illustrates the advantages offered by dissolution; the remainder of the section describes the details of the techniques for speeding up tableau deductions. Section 4 discusses factoring and the class of formulas on which analytic tableau methods were first shown to be intractable.

We describe only the propositional case here. The speedup theorems and the way in which dissolution generalizes the tableau method also apply to the structure of the proof tree in first order tableau operations. Proofs are omitted here but may be found in [18].

† This research was supported in part by the National Science Foundation under Grants CCR-9101208 and CCR-9202013.

2. Path Dissolution and Analytic Tableaux

Path dissolution operates on a link within a formula by restructuring the formula in such a way that all paths through the link vanish. The tableau method restructures a formula so that the paths through the link are immediately accessible and then marks them closed, in effect deleting them. It does this by selectively expanding the formula towards disjunctive normal form. As we shall see, many literal duplications resulting from these operations can be avoided with the introduction of dissolution steps. A first speedup is achieved by avoiding certain duplications in a tableau deduction without essentially altering the deduction. Further speedups are obtained by the omission of duplications that may alter the deduction.

The path-deletion strategy that both techniques employ results in *strong completeness* at the propositional level: Any sequence of steps will eventually produce a linkless formula, and, in particular, in the case of an unsatisfiable formula, will produce the empty formula. This property means that both techniques are well suited for situations in which one is interested in finding the satisfying interpretations – models – of a formula. Many deduction methods such as resolution, which may be useful in the automated theorem proving setting, may not be effective for finding models. In this paper we will therefore primarily use the term *deduction*, rather than *proof*, when discussing applications of either dissolution or analytic tableaux. (We will not substitute *deduction tree* for *proof tree*.)

2.1. Semantic Graphs

Formulas in this paper will be represented graphically in a manner that can be easily understood by considering a simple example. In Figure 1, the formula on the left is displayed graphically on the right:

$$((\overline{C} \wedge A) \vee D) \wedge (\overline{A} \vee (B \wedge C)) \quad \equiv$$

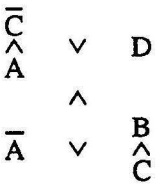


Figure 1

Disjunctions are displayed horizontally, conjunctions vertically. Note that the formula is in *negation normal form* (NNF): The only logical connectives used are AND and OR, and the negations are at the atomic level. A formula so represented is called a *semantic graph*. Essentially, the only difference between a semantic graph and a formula in NNF is the point of view, and we will use either term depending upon the desired emphasis. For a more detailed exposition, see [13]; we urge the reader to consider this example carefully and to refer back to it as necessary.

The graph above contains four *c-paths* (maximal conjunctions of literals): $\{\overline{C}, A, \overline{A}\}$, $\{\overline{C}, A, B, C\}$, $\{D, \overline{A}\}$, $\{D, B, C\}$. A formula is unsatisfiable if and only if every c-path is unsatisfiable, and a c-path is unsatisfiable if and only if it contains a link (a complementary pair of literals). The idea of path dissolution is to eliminate all paths through a given link. Repeated applications produce a linkless formula. Hence all remaining paths are satisfiable, so the original formula is unsatisfiable if and only if the fully dissolved formula is empty.

2.2. Path Dissolution

We can get an intuitive idea of how dissolution works by dissolving on the link $\{A, \overline{A}\}$ in Figure 1; the dissolvent is shown in Figure 2.

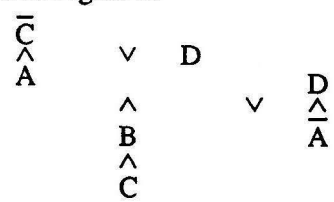


Figure 2

This equivalent graph contains three c-paths – the one path through the link is no longer present.

In general path dissolution is applicable to collections of links; here we restrict attention to single links. If $\{A, \overline{A}\}$ is the link under consideration, suppose A and $\overline{A}$ reside in conjoined subgraphs X and Y, respectively. The *c-path complement* of A with respect to X, written $CC(A, X)$, is the subgraph of X consisting of all literals in X that lie on paths that do not

contain A; the *c-path extension* of A with respect to X, written CPE(A, X), is the subgraph containing all literals in X that lie on paths that *do* contain A. Intuitively, the paths through $(X \wedge Y)$ that do not contain the link are those through $(X \wedge \mathrm{CC}(\bar{A}, Y))$ plus those through $(\mathrm{CC}(A, X) \wedge \mathrm{CPE}(\bar{A}, Y))$. The *dissolvent* of the link $\{A, \bar{A}\}$ in the subgraph $H = X \wedge Y$, denoted DV($\{A, \bar{A}\}$, H), is defined to be

$$\begin{array}{ccc} X & & \mathrm{CC}(A, X) \\ \wedge & \vee & \wedge \\ \mathrm{CC}(\bar{A}, Y) & & \mathrm{CPE}(\bar{A}, Y) \end{array}$$

See [14] for more detail and for the formal definitions.

The reader may have observed that this definition is not symmetric with respect to X and Y. In fact, there are three possible definitions of the dissolvent: the one given, one in which the roles of X and Y are reversed, and one that is symmetric. The graphs produced by the different formulations are distinct, but their respective sets of c-paths are identical: all original c-paths that do not contain both A and $\bar{A}$. The definition given is the one used in our dissolution-based system (the system requires that Y be the larger of the two, swapping if necessary).

A major expense in many theorem proving systems is duplication of literals; this is certainly the case with both dissolution and the tableau method. The cost of duplication may be reduced with techniques such as structure sharing; duplicating a subformula can sometimes be accomplished by merely duplicating a pointer. This may be quite valuable to an implementor, but it does not affect the speedup theorems presented here, which *eliminate* duplications (and corresponding closures) from tableau deductions.

The source of all duplications when dissolving can be inferred from the definition of dissolution and from some simple properties of CC and CPE: Since $\mathrm{CC}(A, X) \subset X$, the nodes comprising CC(A, X) are duplicated, and since $\mathrm{CPE}(A, Y) \cup \mathrm{CC}(A, Y) = Y$, the nodes in $\mathrm{CPE}(A, Y) \cap \mathrm{CC}(A, Y)$ are also duplicated. Duplications in the tableau method result from *dispersing* (defined below) subformulas to the ends of tree paths.

A special case of dissolution, important for the analysis that follows, arises when X consists of A alone; then CC(A, X) is empty, and the dissolvent of the link $\{A, \bar{A}\}$ in the subgraph $X \wedge Y$ is $\begin{array}{c} X \\ \wedge \\ \mathrm{CC}(\bar{A}, Y) \end{array}$. Observe that the effect of dissolving is to replace Y by $\mathrm{CC}(\bar{A}, Y)$, and that $\mathrm{CC}(\bar{A}, Y)$ is formed by deleting $\bar{A}$ and anything directly conjoined to it. Hence no duplications whatsoever are required. We refer to a subgraph such as $A \wedge W \wedge Y$, where A occurs in Y, as a *unit subgraph*, and refer to its dissolvent as a *unit dissolvent*. If, in addition, Y is an empty intersection subgraph of A, we call $A \wedge W \wedge Y$ a *tableau subgraph* and refer to its dissolvent as a *tableau dissolvent*.

2.3. Analytic Tableaux

A basic familiarity with the tableau method as described by Smullyan [21] will be helpful, though the description presented here is self contained. There is a natural correspondence between tableau proof trees and semantic graphs. As a result, in order to facilitate the description of the manner in which dissolution generalizes tableaux, we cast the tableau technique in terms of semantic graphs.

The general tableau method allows for formulas containing implications and negations at any level. Such formulas can be converted to NNF in linear time. The general method also allows for closing a path containing a "link" consisting of an arbitrary subformula $\mathcal{F}$ and its syntactic complement $\neg\mathcal{F}$. Implementors often ignore even this special case of non-atomic complementarity, but such "non-atomic links" can be noted via simple preprocessing before conversion to NNF and subsequently employed by both tableau- and dissolution-based methods. Nonetheless, this is an unimportant case, and for the remainder of this paper we restrict attention to negation normal form.

The tableau method is often described with signed formulas, but, as Smullyan points out, this is unnecessary: The initial formula is implicitly signed true and tested for consistency. The description in terms of semantic graphs then requires three rules: *separation*, *dispersion*, and *closure*. It is also convenient to designate certain subgraphs as *primary*; they form a tree that corresponds precisely to the proof tree maintained by the tableau method.

Initially, the entire graph is the only primary subgraph. A separation is performed on any primary subgraph whose highest level connective is a conjunction by removing the primary

designation from it and bestowing that designation on its conjuncts. This corresponds to Smullyan's α-rule. There is essentially no cost to this operation, and it may be regarded as automatic whenever a conjunction becomes primary.

A separation may also be performed on a disjunction that is a leaf in the primary tree. (Separating an interior disjunction is not allowed since such an operation would destroy the tree structure.) Separations of such disjunctions should not be regarded as automatic: This operation increases the number of paths in the tree (we call such paths *tree* paths to distinguish them from c-paths), so, although there is no cost to the operation itself, there is a potential penalty from the extra paths.

Automatically separating conjunctions has the effect of treating conjunction as an n-ary operator, and in effect the traditional tableau approach so treats conjunction. One result is that commuting conjuncts that lie along a single tree path (with no intermediate branch points) is inconsequential. Not automatically separating disjunctions is consistent with the typical tableau approach of treating disjunction as a binary operator. We will follow both of these conventions – n-ary for conjunction, binary for disjunction – in this paper, although the results apply with any representation.

The process of dispersing a primary subgraph whose highest level connective is a disjunction may now be defined precisely: A copy of the subgraph is placed at the end of one path descending from it and separated[1]. This operation corresponds to Smullyan's β-rule. For example, suppose that $X = X_1 \vee X_2$ is a primary subgraph, and that the leaf Y is a descendent of X. On the left, we show the original tree path from X to Y; on the right is the result after X has been dispersed.

The subgraphs X and Y remain primary, and the copies of X_1 and X_2 are now designated as primary. Note that if a subgraph is eventually dispersed to the ends of every path descending from it, the original copy is no longer required. As a result the last copy may be effected by simply moving the subgraph. We will use the term *trivial dispersion* for a dispersion that requires no literal duplications.

The key operations in a tableau deduction are the closures, which close tree paths. Marking a tree path closed is equivalent to deleting it; in this paper we will describe closures as deletions. In terms of the tree of primaries, a tree path may be closed when a separation or a dispersion makes primary a literal that forms a link with one of its ancestors. The primary literal and all of its descendants are deleted, and any leaves that result are in turn deleted. Since the tree path through the link is removed, the effect is to delete all c-paths through the link.

As an example, the graph in Figure 3 represents the unsatisfiable formula $(((A \wedge B) \vee C) \wedge C \wedge (A \vee B))$. Boxes are used to designate primary subgraphs; initially the entire graph is the only one. Since it is a conjunction, this primary subgraph is automatically separated, and Figure 4 results.

In Figure 4, there is yet only one path in the tree. As a result, a dispersion may be performed by moving (without any duplication) any primary to the leaf position and then separating. For simplicity, we separate the primary that is already a leaf. As a semantic graph, the resulting formula in Figure 5 is unchanged. But by designating A and B as primary, the proof tree has split and now contains two paths.

---

[1] There is another common view (for example, see [7,19]) of the tableaux method in which the dispersed subgraph is moved to the ends of *every* tree path descending from it. This allows, in the propositional case, the *removal* of the original copy of the dispersed subgraph. The results in this paper apply with that description; see [15,16,17] for the details.

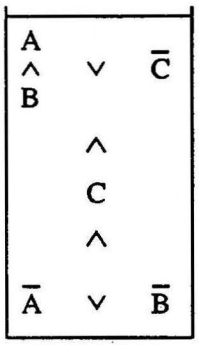


Figure 3.

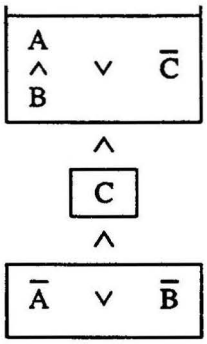


Figure 4.

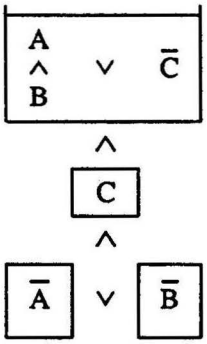


Figure 5.

Only one operation can follow: dispersion of the upper primary. In Figure 6, it is dispersed twice, once for each tree path, allowing the deletion of the original copy at the root.

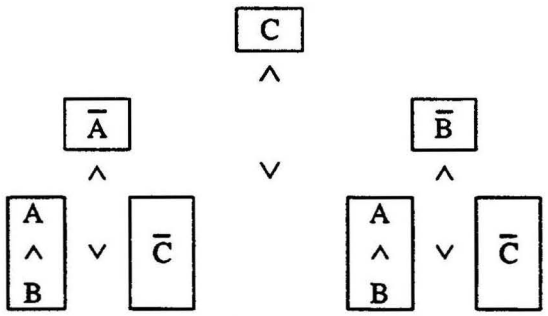


Figure 6.

The remainder of the proof is straightforward. Each of the primary subgraphs ($A \wedge B$) is automatically separated into two primaries, and each of the four paths in the tree can then be closed; two paths contain a $\{C, \overline{C}\}$ link, another contains $\{A, A\}$, and the fourth contains $\{B, B\}$.

It is worth noting that any tree path from the root to a leaf may be thought of as a collection of c-paths. For instance, the single tree path in Figure 5 that contains $\overline{A}$ corresponds to the c-paths $\{\{A, B, C, \overline{A}\}, \{\overline{C}, C, \overline{A}\}\}$. Note also that dispersion is the source of all literal duplication in the tableau method; in Figure 6, an extra copy (in this case only one) of $((A \wedge B) \vee \overline{C})$ has been created for all but the last descendent leaf to which it has been dispersed.

2.4. Path Dissolution as a Generalization of Analytic Tableaux

A separation has no effect on a semantic graph; it is a bookkeeping device employed to keep track of the primary subgraphs. A dispersion involves only standard laws of propositional logic, which of course can be used with any logical system. Lemma 1 states that a closure is a special kind of dissolution step. Conclusion: Dissolution is a generalization of the tableau method.

Lemma 1. A closure operation may be regarded as a tableau dissolution step.

Theorem 1. Path dissolution generalizes the method of analytic tableaux: All tableau deductions are dissolution deductions, but the converse does not hold.

2.5. Analyticity

The traditional view of an *analytic* proof system is one in which any new formula introduced is a copy of one that is present as an explicit subformula of the original. Viewing a tableau deduction as a proof tree satisfies this condition: Each node in the proof tree is a subformula of the original. With the view presented here – that the entire proof tree is a single NNF formula – analytic tableau deductions are not analytic! As an example, the tree of Figure 6 is, as a single formula, clearly absent from Figures 3, 4, and 5. However, each of the primary subgraphs of Figure 6 does occur as a subformula of the original formula. As a result, in this paper tree structures will be called analytic if each node (primary subgraph) in the tree is an explicit subformula of the original formula.

Path dissolution in its full generality is not analytic. For example, the formula in Figure 2 contains the subformula $D \wedge A$, which is not an explicit subformula in Figure 1.

Certain of the speedup results (Theorems 2 and 4) preserve analyticity and others do not. This may be of interest to the implementor since analytic proof procedures allow for substantial structure sharing.

3. Improving Tableau Deductions with Path Dissolution

Closure steps typically follow dispersions that open up a link. We define a *non-trivial closure-enabling dispersion* to be one in which a closure is made possible by dispersing a primary subgraph that has at least two descendent leaves on paths along which it has not yet been dispersed. Another way to view a non-trivial dispersion is as one in which the original copy cannot be deleted after it is dispersed. We single out these dispersions because – see Theorem 2 below – they may be made more efficient with applications of path dissolution. (Trivial dispersions cannot be speeded up since they can be accomplished without any duplications.)

3.1. Speeding Up Non-Trivial Dispersions

The next example illustrates the essentials of both the proof, which is omitted here, and the meaning of the first speedup theorem. The two steps described cover all possible cases; in fact, the example virtually constitutes a proof. The theorem is of practical value to the serious implementor[2].

Consider the primary tree in Figure 7 (the box notation has been dropped). There are four tree paths; the only non-literals are $Y = \overset{A}{\underset{U}{\wedge}} \vee W$ and $Y' = \overset{B}{\underset{U'}{\wedge}} \vee W'$; and Y, Y', A, B, and the L_i's are all primary.

Suppose that the next step in the tableau deduction is the dispersion of Y along the path terminating in L_1. Since $Y = \overset{A}{\underset{U}{\wedge}} \vee W$, this dispersion is followed by the separation of $\overset{A}{\underset{U}{\wedge}}$ from W. The original Y cannot be deleted since it has not yet been dispersed to all of its descendent leaves. The tree paths increase from four to five, and one of these paths may be closed because of the link $\{A, \bar{A}\}$. The closure is indicated by $\boxtimes$, and the graph in Figure 8 is the result.

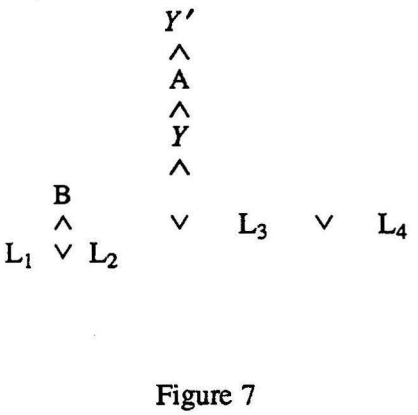


Figure 7

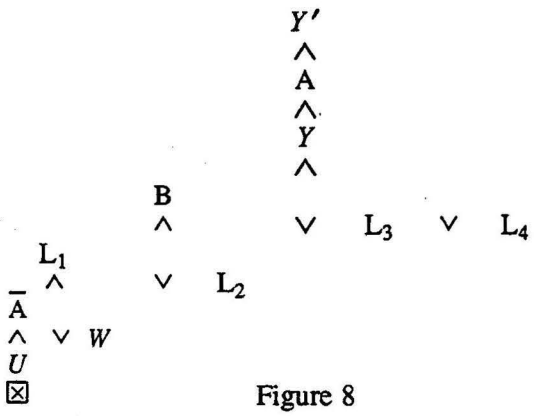


Figure 8

A careful inspection of Figure 7, however, reveals that the subgraph $\overset{A}{\underset{Y}{\wedge}}$ is a tableau subgraph. If we were to *dissolve* on the link $\{A, \bar{A}\}$, the result would be $\overset{A}{\underset{CC(\bar{A}, Y)}{\wedge}}$, which would amount to deleting $\overset{\bar{A}}{\underset{U}{\wedge}}$ and would produce the graph in Figure 9.

Copying W (without separating) to the end of the leftmost tree path in Figure 9 will yield the graph of Figure 8, except that the original Y in Figure 8 is replaced by W in Figure 9, in

[2] Reiner Hähnle has informed us that he did achieve a significant speedup in his tableau-based system by applying the theorem.

effect deleting the closed path. This is necessarily an advantage; the justification can be found in the full paper [18]. More importantly, the creation and deletion of a copy of $\overset{\overline{A}}{\underset{U}{\wedge}}$ has been saved.

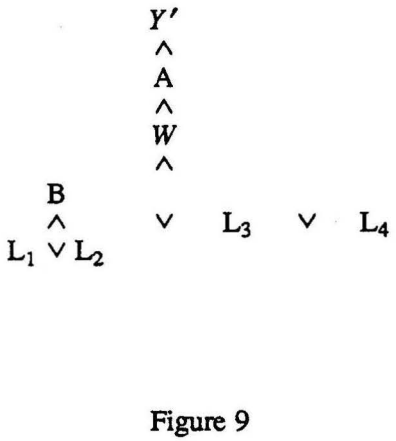


Figure 9

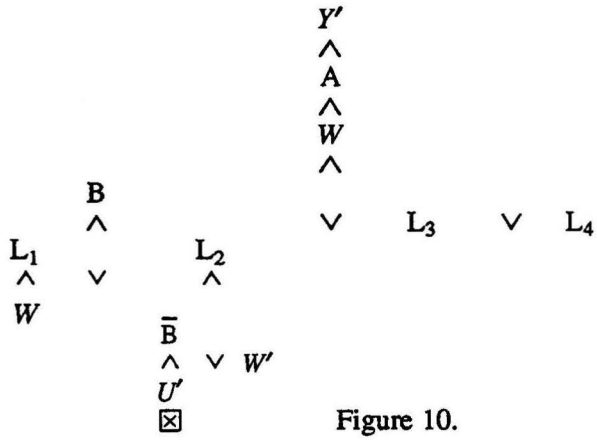


Figure 10.

The dispersion of W to the end of the leftmost path in Figure 9 may seem wasteful; it is done to ensure that the trees produced by the two methods are essentially identical – the only difference is that the node W in the dissolved tree is a subgraph of the corresponding node Y in the other tree. Theorem 4 in Section 3.3 states that these *excess dispersions* may in fact be eliminated. Observe that analyticity is preserved whether or not W is dispersed: W must be an explicit subgraph of the original formula if Y is.

Continuing from Figure 9 after copying W below L_1, suppose the next step disperses $Y' = \overset{B}{\underset{U'}{\wedge}} \vee W'$ along the path ending with L_2, opening up the link $\{\overline{B}, B\}$. After closing on that link, the result is the graph of Figure 10.

Using dissolution to speed up this step requires some care; there is no tableau subgraph for $\{\overline{B}, B\}$ in Figure 9. Observe that if Y' were dispersed to a point just below B in the primary tree, the result would be the graph in Figure 11. (Recall that W was copied to the end of the leftmost path in Figure 9.) We could then dissolve on the link $\{B, \overline{B}\}$, producing Figure 12.

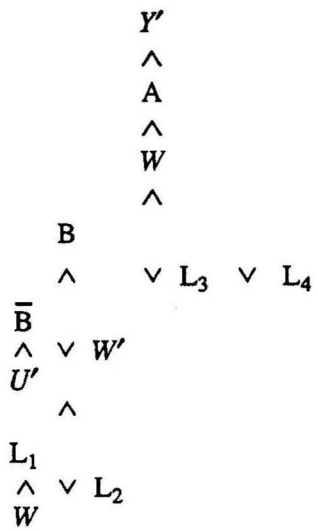


Figure 11.

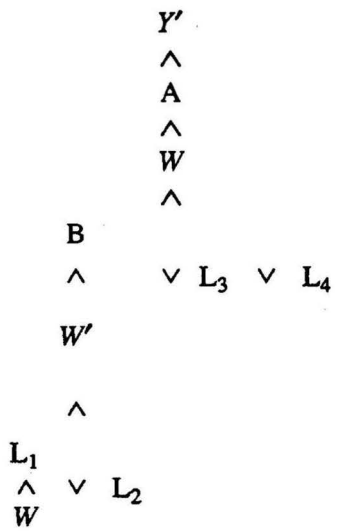


Figure 12.

The key to obtaining the desired speedup is to observe that actually placing a copy of Y' below B is unnecessary: Knowing what the dissolvent will be allows the direct placement of a copy of W' on the path below B. In the tableau deduction, a copy of W' remains at the end of the path below L_2 after a copy of Y' is dispersed and separated. All that is required to speed up the tableau deduction is to place W' at the end of the path, rather than just below B, and

this amounts to the graph of Figure 10 after the closure enabled by the $\{\bar{B}, B\}$ link.

By predicting the dissolvent, W' could be placed just below B, as in Figure 12, rather than at the end of the path, as strict adherence to the tableau method requires (Figure 10). The effect would be to capture a second dispersion of Y' and subsequent closure along the leftmost path. However, we place W' at the end of the path to ensure that the dissolution-aided tableau proof tree has the structure of the original tableau proof tree.

Theorem 2 below states that as a tableau proof tree develops, any non-trivial closure-enabling dispersion can be speeded up with the application of path dissolution technology. At each step, although the proof tree is structurally identical to the one produced without dissolution, the corresponding nodes may differ, as the previous example illustrates. The two proof trees can be kept identical. In Figure 9 of the example the result W of the tableau dissolution need not replace Y in the proof tree. Having computed W, it may be dispersed to the end of the designated path, but Y may be left unchanged. The creation and deletion of one tree path is still saved, but the resulting "modified" proof tree is identical to the unmodified tree. With this variation, Theorem 2 amounts to a look-ahead strategy whereby tableau dissolvents are predicted and then dispersed, and the corresponding tree path extensions and closures are simply avoided. This observation may be important in the first order case: Activation of the link will typically require Y to be instantiated, and the original Y in full generality may still be required to complete the proof.

Theorem 2. The number of literal duplications in each non-trivial closure-enabling dispersion in a tableau deduction can be reduced with the application of dissolution technology.

In the dispersion of Y in the previous example, certain seemingly unnecessary dispersions are discussed; Theorem 4 states that they are indeed unnecessary.

For the dispersion of Y' in the example, it would appear to be an advantage to place a copy of W' just below A, rather than at the end of θ. Doing so would have the effect of dispersing Y' along all paths descending from A and subsequently closing those paths that contain A. Moreover, these dispersions are accomplished with no literal duplications whatsoever. However, such placement of W' would alter the structure of the modified tree, and it would be difficult to verify that the speedup would not be lost later in the proof.

Theorem 2 has practical value: It can be implemented in a straightforward manner. A typical implementation will employ a *next-link strategy*: Choose a link on which to perform a closure, and then select dispersions to open up that link. If $\{A, \bar{A}\}$ is the link, as soon as one literal, say A, becomes primary, one can check whether the subgraph containing $\bar{A}$ is an empty intersection subgraph of $\bar{A}$. If it is, depending on the location of that subgraph, Case 1 or Case 2 of the theorem must apply.

Theorem 2 may apply more often at the first order level: Instantiation may make it necessary to copy rather than to move primaries during a dispersion, and this may make that dispersion non-trivial even though it would have been trivial in the propositional case.

3.2. Speeding Up Trivial Dispersions

The reader may wonder how often closures are enabled by non-trivial dispersions. Certainly not always: In Section 4.2 we show that, with factoring, the tableau method can produce proofs for a certain class of formulas without duplicating a single literal. Typical tableau deductions, however, require many non-trivial dispersions. Perhaps surprisingly, closures that require non-trivial dispersions may nevertheless be enabled by trivial dispersions. This can happen if, for example, $\{A\}$ becomes primary, and $\bar{A}$ is buried deeply in its primary subgraph. The dispersion that made $\{A\}$ primary may or may not have been trivial, but it will place A below $\bar{A}$ in the tree and create at least two tree paths below A. This forces the next dispersion of the primary containing $\bar{A}$ to be non-trivial. Several additional dispersions may then be required to enable the $\{A, \bar{A}\}$ link, and these may all be trivial.

Dissolution offers a speedup in these situations. To see how, let $\{A, \bar{A}\}$ be an arbitrary link whose closure requires non-trivial dispersions but is enabled by a trivial dispersion, and let P_A and $P_{\bar{A}}$ be defined as above. (Dispersions of subgraphs other than these cannot contribute to enabling the closure, and we ignore them.) Clearly, until either $P_A = \{A\}$ or $P_{\bar{A}} = \{A\}$, trivial dispersions alone cannot enable the closure. So assume that a dispersion produces $P_A = \{A\}$; i.e., assume that $\{A\}$ has become primary. Since P_A was just dispersed, $P_{\bar{A}}$ is above A in the tree. Also, since this dispersion made $\{A\}$ primary, P_A must have been a disjunction before the dispersion; i.e., this dispersion must create at least two tree paths. Thus,

at least one more dispersion of $P_{\bar{A}}^{-}$ is required to enable the closure, and the first such is non-trivial. After that dispersion, A (and hence $P_{\bar{A}}^{-}$) is below A, which is primary, and any remaining dispersions of $P_{\bar{A}}^{-}$ required to enable the closure are trivial.

The _unit dissolvent of the $\{A, \bar{A}\}$ link in the unit subgraph $\{A, P_{\bar{A}}^{-}\}$ is $(A \wedge \underline{\wedge} CC(\bar{A}, P_{\bar{A}}^{-}))$. The tableau steps can be accomplished, as in the example by dispersing $CC(\bar{A}, P_{\bar{A}}^{-})$, saving duplication of the deleted literals. This gives us

Theorem 3. Path dissolution offers a speedup to a tableau deduction in which there is a closure that requires non-trivial dispersions but which is enabled by a trivial dispersion. □

3.3. Implementation Considerations

If one examines the example for Theorem 2, it is clear that the improvements can easily be added to any tableau prover. In practice, an implementor is unlikely to program his prover to perform those seemingly unnecessary dispersions that are used to avoid altering the structure of the tableau deduction. Theorem 4 in essence justifies this approach for the case involving the tableau subgraph: It shows that given any (dissolution-aided) tableau deduction with those dispersions (called the *excess* dispersions in the theorem), there is a faster one without them. This modification will induce a deduction that in general is not obtainable using pure tableau methods. Nevertheless, the new deduction will be analytic and will employ primary trees.

Theorem 4. Given a formula $\mathcal{F}$ and a tableau deduction of $\mathcal{F}$ that has been speeded up with applications of Theorem 2, there is a deduction of $\mathcal{F}$ that is at least as efficient as the original, and that contains none of the excess (Case 1) dispersions.

4. Intractability

In this section we discuss the performance of both the tableau method and path dissolution on a class of formulas introduced by Cook and Reckhow. A key ingredient is the use of factoring, by which we mean applying the distributive laws of logic to formulas so as to combine multiple occurrences of subformulas. For example, $(A \vee B) \wedge (A \vee C)$ factors (conjunctively) to $A \vee (B \wedge C)$. With logical formulas, since two distributive laws hold, both conjunctive and disjunctive factoring can be done. In the next section, we will be concerned primarily with conjunctive factoring.

4.1. Factoring

An automated deduction system that employs clause form cannot factor formulas since the factored formula will not in general be in clause form. On the other hand, any technique that uses NNF can factor, and, if the technique is path based, factoring may improve performance since it (often substantially) reduces the number of c-paths. In particular, both the tableau method and path dissolution are likely to benefit from factoring. We added factoring to Dissolver, our dissolution-based system, early in its development, and a dramatic speedup was achieved on *every* propositional formula that we input in clause form. (The system was run primarily on unsatisfiable formulas, and factoring is almost always possible with unsatisfiable formulas in clause form.)

Our experiments with factoring indicated that the most efficient way to use (conjunctive) factoring was to factor as much as possible as a preprocessing step and then never to factor again. Intermediate factoring did not hurt per se, but it did not help, and, as the formula grew during a deduction, the time spent trying to factor became significant.

Disjunctive factoring may actually have a negative impact on performance. The reason is that "inverse disjunctive factoring" (i.e., multiplying) can "open up" a link in such a way as to reduce the number of literals duplicated when dissolving on the link. Indeed, it is exactly this behavior that allows the tableau method to work at all: Tableau dispersions are a kind of "partial multiplication."

It is interesting to observe that factoring "subsumes" subsumption. For example:

$$A \wedge (A \vee B) = (A \vee \text{false}) \wedge (A \vee B) = A \vee (\text{false} \wedge B) = (A \vee \text{false}) = A .$$

This analysis obviously extends to any pair of propositional disjunctions in which one subsumes the other.

Factoring depends only on distributivity, a basic law of logic, and we do not regard the use of factoring as any more of an enhancement to path dissolution than associativity, commutativity and merging are to resolution. With analytic tableaux it is somewhat different. The

traditional view of a labeled tree does not naturally admit factoring during a tableau deduction, but factoring the input formula is certainly reasonable. The view presented here (an NNF formula in which certain subgraphs form a primary tree) is a much more natural setting in which to do factoring. In the next section, we discuss a class of formulas that Cook and Reckhow [4] have shown to be intractable for analytic tableaux; we show that preprocessing these formulas with factoring allows them to be handled in polynomial (in fact linear) time by analytic tableaux.

4.2. Hard Formulas Made Easy With Factoring

In [4], Cook and Reckhow describe a class of formulas $\{T_m, m \geq 1\}$ that are intractable for analytic tableaux but that can be handled in linear time by resolution. In this section we show that by factoring the input formula, both path dissolution and the tableau method can also handle these formulas in linear time.

This is an improvement of a similar result that we presented in 1990 at the *International Symposium on Symbolic and Algebraic Computation* [17]. There, factoring was applied *during* the proof, i.e., to the proof tree itself, as well as to the input formula. In 1992, Eder [6] modified the tableau calculus to include a factoring operation on the paths of the proof tree. Since tree paths may be regarded as conjunctions of their primary subgraphs, he observed that subsumption may be possible between two of them. He showed that the resulting "tableau calculus with factorization" admits proofs for this class that are linear in $|T_m|$, the number of input clauses.

In most tableau calculi, operations on the proof tree are considered to be part of the deduction machinery. Because we treat the entire proof tree as a formula, the distinction between factoring the original formula and factoring during the deduction is blurred. Nevertheless, we show here that it is necessary only to factor the initial formula and that, unlike in [6] and [17], no modification whatsoever to the standard tableau procedure is necessary. These proofs are also linear in $|T_m|$, but they require no literal duplications.

The notation used to describe the collection $\{T_m\}$ of formulas is from [4]. Consider, for example, T_3 in Figure 13.

$$
\begin{array}{lllll}
P^1 & \vee & P^2_{+} & \vee & P^3_{++} \\
 & \wedge & & & \\
P^1 & \vee & P^2_{+} & \vee & \overline{P^3_{++}} \\
 & \wedge & & & \\
P^1 & \vee & \overline{P^2_{+}} & \vee & P^3_{+-} \\
 & \wedge & & & \\
P^1 & \vee & \overline{P^2_{+}} & \vee & \overline{P^3_{+-}} \\
 & \wedge & & & \\
\overline{P^1} & \vee & P^2_{-} & \vee & P^3_{-+} \\
 & \wedge & & & \\
\overline{P^1} & \vee & P^2_{-} & \vee & \overline{P^3_{-+}} \\
 & \wedge & & & \\
\overline{P^1} & \vee & \overline{P^2_{-}} & \vee & P^3_{--} \\
 & \wedge & & & \\
\overline{P^1} & \vee & \overline{P^2_{-}} & \vee & \overline{P^3_{--}}
\end{array}
$$

Figure 13.

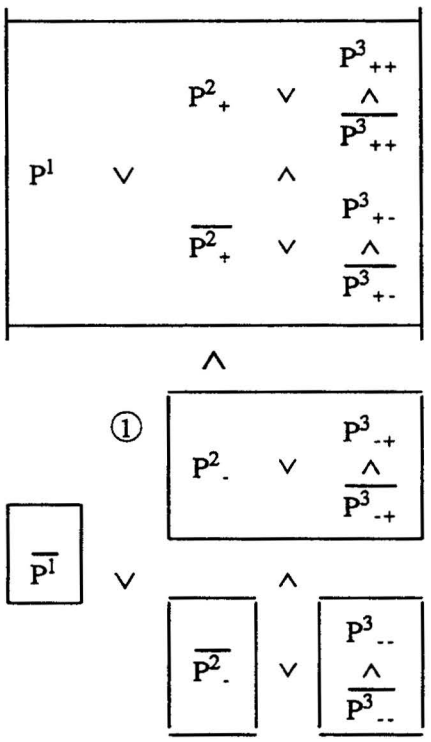


Figure 14.

Each clause contains atoms of the form $P^i_{\#}$, where # is a string of +'s and -'s. The superscripts in each clause always form the sequence 1, 2, ... m. The subscript of each literal is exactly the sequence of signs of the preceding literals in its clause. When T_m is built from T_{m-1}, each $P^m_{\#}$

is added both positively and negatively. It is easy to see that T_m has 2^m clauses, each of which contains m literals.

Cook and Reckhow observed that (without factoring!) analytic tableaux require time exponential in 2^m, the number of clauses in T_m. To see how factoring can lead to linear time tableau proofs of these formulas, again consider T_3. Observe that the first four (for T_m, 2^{m-1}) clauses can be factored on their first literal; the last four clauses can be similarly factored. The result is a conjunction of two disjunctions; the first contains P^1 and four two-literal clauses, and the second contains P^1 and the same four clauses. Further factoring can be done on pairs of clauses containing those literals whose superscripts are 2.

It is easy to see that by factoring on all variables with superscripts less than m, T_m can be completely factored: There are 2^{i-1} distinct variables with superscript i, and each occurs positively 2^{m-i} times and negatively 2^{m-i} times.

For T^3, several separations applied to the completely factored formula produces the graph of Figure 14.

After one more separation on the the conjunction along the right-hand path, a closure can be performed deleting $P^3_{..}$ and $P^3_{...}$. The primary node labeled ① may now be dispersed with deletion – i.e., the dispersion is trivial and the primary may be moved to the end of the one tree path on which it resides. We leave the remaining details to the interested reader.

As it turns out, with exactly one trivial dispersion, we have essentially solved T_2; three such will solve T_3. More generally, if T_i requires D_i trivial dispersions, then there are two subformulas in T_{i+1} isomorphic to T_i, each of which can be handled by D_i trivial dispersions. One additional trivial dispersion will move and separate the upper half of T_{i+1}. Obviously, $D_{i+1} = 2 \cdot D_i + 1$, and, since $D_2 = 1$, we have $D_m = 2^{m-1} - 1$.

This shows that, with initial factoring, this class of formulas can be handled without any duplication whatsoever by analytic tableaux. As a result, the number of closures is exactly the number of links in the fully factored formula, which is easily verified to be $2^m - 1$.

It is straightforward to verify that proofs requiring $2^m - 1$ steps can be produced by dissolution in a similar manner.

Theorem 5. Both path dissolution and the method of analytic tableaux admit polynomial time proofs of the class $\{T_m\}$.

4.3. Conclusions

Cook and Reckhow classify a proof system as *super* if it admits polynomial time proofs for all unsatisfiable formulas. The existence of a super proof system would imply that NP = co-NP. They further classify proof systems as to whether they are known to be non-super. Haken's fine work showing that resolution required exponential time on the pigeonhole formulas [11] came several years after their paper, and it moved resolution into the class, "known to be non-super." The work Cook and Reckhow did with tableau placed it in the same class. Armed with factoring as a preprocessing step, tableau is no longer in that class: We know of no collection of fully factored formulas that is intractable for standard tableau methods (nor, of course, for dissolution). It would be rather surprising if such a class did not exist.

References

1. Beth, E.W. *The Foundations of Mathematics.* North Holland, Amsterdam (1965).
2. Bibel, W. Tautology testing with a generalized matrix reduction method. *Theoretical Computer Science,* **8** (1979) 31-44.
3. Carnielli, W.A. Systematization of finite many-valued logics through the method of tableaux. Journal of Symbolic Logic, 52,2 (June 1987), 473-493.
4. Cook, S. and Reckhow, R. On the lengths of proofs in the propositional calculus. *Proceedings of the Sixth Annual ACM SToC.* May, 1974, 135-148.
5. Doherty, P. Preliminary report: NM3 – A three-valued non-monotonic formalism. *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems,* Knoxville, Tennessee October 25-27, 1990. In Methodologies for Intelligent Systems, **5** (Ras, Z., Zemankova, M., and Emrich, M. eds.) North-Holland, 1990, 498-505.
6. Eder, E. *Relative Complexities of First Order Calculi,* Friedr. Vieweg & Sohn, 1992.
7. Elfrink, B. and Reichgelt, H. The use of assertion-time inference in logic-based knowledge bases. *Proceedings of ECAI-88,* Munich, W. Germany, 1988
8. Fitting, M. *First Order Logic and Automatic Theorem Proving,* Springer-Verlag, 1990.
9. Gentzen, G. *Investigations in Logical Deduction.* in Szabo 1969, 132-213.
10. Hähnle, R. Uniform notation of tableau rules for multiple-valued logics. *Proceedings of the 21st International Symposium on Multiple-Valued Logic,* Victoria, B.C., Canada, May 26-29, 1991, 238-245.
11. Haken, A. The intractability of resolution. *Theoretical Computer Science,* 39 (1985), 297-308.
12. Hintikka, J. Form and content in quantification theory. *Acta Philosophica Fennica* **8,** Helsinki (1955).
13. Murray, N.V., and Rosenthal, E. Inference with path resolution and semantic graphs. *J.ACM* 34,2 (April 1987), 225-254.
14. Murray, N.V., and Rosenthal, E. Dissolution: making paths vanish. To appear, *J.ACM.*
15. Murray, N.V., and Rosenthal, E. Employing path dissolution to shorten tableau proofs. *Proceedings* of the *1989 International Symposium on Symbolic and Algebraic Computation,* Portland, Oregon July 17-19, 1989, 373-381.
16. Murray, N.V., and Rosenthal, E. Improving tableau proofs. *Proceedings* of the *Fourth International Symposium on Methodologies for Intelligent Systems,* Charlotte, North Carolina October 11-14, 1989. In Methodologies for Intelligent Systems, 4 (Ras, Z. ed.) North-Holland, 1989, 477-484.
17. Murray, N.V., and Rosenthal, E. Reexamining Intractability of Analytic Tableaux. *Proceedings* of the *1990 International Symposium on Symbolic and Algebraic Computation,* Tokyo, Japan, Aug. 20-24, 1990, 52-59.
18. Murray, N.V., and Rosenthal, E. On the relative merits of path dissolution and the method of analytic tableaux. Technical Report TR 93-5, Dept. of Computer Science, SUNY at Albany.
19. Oppacher, F. and Suen, E. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning* 4, 69-100, (1988).
20. Schwind, C. A tableau based theorem prover for a decidable subset of default logic. *Proceedings* of the *10th International Conference on Automated Deduction,* Kaiserslautern, W. Germany, July 24-27, 1990. In *Lecture Notes in Computer Science,* Springer-Verlag, Vol. 449, 528-542.
21. Smullyan, R.M. *First-Order Logic.* Springer Verlag, 1968.
22. Wallen, L. *Automated Deduction in Nonclassical Logics.* MIT Press, Cambridge, MA, 1989.

A framework for integrating equality reasoning into the extension procedure[1]

Uwe Petermann
Inst. f. Informatik, Universität Leipzig
Augustuspl. 10, D-O-7010 Leipzig (Germany)
Net: peterman@informatik.uni-leipzig.de

Abstract

We introduce a calculus which allows the integration of different strategies of equality reasoning into the connection method. We show that the calculus which we introduce may be instantiated to paramodulation, RUE-like derivations and to relaxed paramodulation. As a formal tool we use the pool calculus extending it in order to treat equality reasoning as an instance of partial theory reasoning. The work reported in this paper aims in the construction of an experimental tool for equality reasoning. We hope that experimental results will give intuitions for the clarification of theoretical problems.

1 Introduction

In the present paper we study possibilities of the integration of methods for equality handling into calculi which are based on the connection method. We focus on the pool calculus [NS91] which is close to the extension procedure [Bib82]. Till now calculi of this class don't support equality handling sufficiently. On the other hand, they represent an interesting alternative to resolution like calculi. They allow non-normalform proving and and take advantage from informations about the formula structure. This raises the wish to extend the applicability of those calculi to equality reasoning. Let us start with a brief overview about equality handling in other calculi.

At the very beginning of first order theorem proving the paramodulation rule [RW69] has been defined within the resolution framework. Much effort has been devoted to pruning the enormous search space which is entered by the this rule. We distinguish three directions: (1) to make the application of the paramodulation rule more goal directed [Mor69, DH86], (2) to use order restrictions [BG92, Rus91], what seems to be more successful, at least by now, (3) to combine paramodulation or its modifications with restrictions of the base calculus [SL91]. Equality reasoning in non-resolution calculi follows more or less the resolution pattern. For an overview see [BFP92].

An alternative approach may be found in [GNPS92]. There, using Stickel's terminology [Sti85], a total theory reasoning approach has been presented. This means that a single inference step relies on the detection of a literal set which is complementary in the theory of equality. Differently to [GNPS92] all of the previously mentioned approaches are, again in Stickel's terminology, instances of partial theory reasoning. In those calculi literal sets which are complementary in the theory of equality are not detected in a single inference step. Rather they are approximated by a number of inference steps. In each such step the actual goals are transformed a bit untill a complementary literal set has been obtained.

[1]This research has been supported by a grant of the Alexander von Humboldt-Stiftung and the Alfried Krupp von Bohlen und Halbach-Stiftung.

Taking in account the plenty of material about equality reasoning one should answer at first to the question which of the already known methods might be carried over to connection calculi. This question is not trivial. Promising resolution calculi with equality handling which rely on paramodulation impose restrictions on the inference rule. The extension procedure may be viewed as a calculus with another restriction of the applicability of the resolution rule. Thus, we cannot expect that a combination of both restrictions gives automatically a complete calculus. This is field for further research.

The calculus presented below is thought as general framework which may be instantiated to several known calculi of equality reasoning which rely on partial equational reasoning. Its implementation can serve as a testbed for several strategies restricting the paramadulation rule. Those different strategies can be obtained by redefining a small number of modules which co-operate with the main inference engine by a well defined interface. An implementation which is based on a PTTP-like prover is ongoing.

2 Preliminaries

We assume that the reader is familiar with the basic notions of first-order logic in clausal form (cf. [Lov78]). For simplifying matters we assume that the equality predicate is the unique predicate of the considered language. If t is a term and α is an address (in other words: an occurrence) in t then by t/α we denote the subterm occurring at that address. For any term s by $t[\alpha \leftarrow s]$ we denote the term which is obtained from t by substituting the occurrence of the subterm at address α by s. An equation is an atomic formula of the form $u \doteq v$; by $u \approx v$ we refer ambigously to either $u \doteq v$ or $v \doteq u$; its negation will be denoted by $u \not\approx v$. Multisets of literals will be denoted as sequences of their elements $L_1, \ldots, L_n$. We will distinguish paths and clauses as two kinds of multisets of literals. We consider a *clause* $L_1, \ldots, L_n$ to be the abbreviation of the universal closure $\forall (L_1 \vee \ldots \vee L_n)$ of the disjunction of its elements. Clauses will be abbreviated also by $\Gamma, \Delta, \Lambda, \Theta$. Γ, Δ means the union $\Gamma \cup \Delta$, Γ, L means $\Gamma \cup \{L\}$ etc. Paths will be abbreviated also by p, q. A *matrix* is a multiset of clauses. A *(partial) path (in) through* a matrix M contains (at most) exactly one literal from each clause of M. A set of paths in matrix M is called a *mating* in M. The meaning of a matrix $\{C_1, \ldots, C_n\}$ is the conjunction $\forall C_1 \wedge \ldots \wedge \forall C_n$. A mating U in a matrix M is called *spanning* if every path through M contains an element of U as subpath. Let E be a (possibly empty) set of positive equality literals. Then any literal set of the form $E, s \not\approx t$ will be called an *eq-connection.*

Let $\mathcal{T}$ be a set of clauses. A $\mathcal{T}$-model is a Herbrand interpretation satisfying $\mathcal{T}$. A syntactic entity S of the mentioned above kind is *$\mathcal{T}$-satisfiable* if there is a $\mathcal{T}$-model satisfying S and *$\mathcal{T}$-unsatisfiable* else. Let $\mathcal{E}$ be the theory of equality, i.e. the clause set consisting of clauses expressing reflexivity, symetry, transitivity and functional and predicative substitutivity. Then the $\mathcal{E}$-models in a given signature are exactly the graphs of the congruence relations in the Herbrand universe.

An eq-connection is called *$\mathcal{E}$-complementary* if its existential closure is $\mathcal{E}$-unsatisfiable. If $\sigma(E, s \not\approx t)$ is $\mathcal{E}$-complementary then the substitution σ is called an *eq-unifier* of the eq-connection $E, s \not\approx t$. **Remark:** It may be seen easily that an eq-unifier σ of an eq-literal $E, u \not\approx v$ is a rigid E-unifier of the term pair $\langle u, v \rangle$ in the sense of [GNPS92]. The notion eq-connection has been defined in [Bib82].

If L is a positive literal then $\bar{L}$ denotes the literal $\neg L$. If L has the form $\neg K$ then $\bar{L}$ denotes the literal K. If p is the path $L_1, \ldots, L_n$ then $\bar{p}$ denotes the clause $\bar{L_1}, \ldots, \bar{L_n}$. And, vice versa, if Γ is the clause $L_1, \ldots, L_n$ then $\bar{p}$ denotes the path $\bar{L_1}, \ldots, \bar{L_n}$. The set

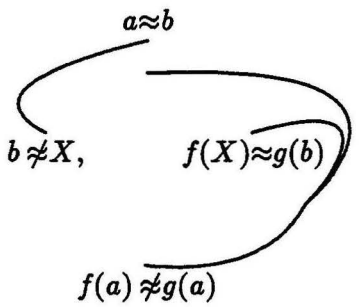
of variables occurring in a term t, literal L, clause Γ or path p will be denoted by $Var(t)$, $Var(L)$, $Var(\Gamma)$ or $Var(p)$ respectively.

3 Pool calculus

The pool calculus [NS91] is one of the formalizations of the connection method [Bib82]. We generalize this calculus in order to obtain a framework for several variants of equational reasoning. In order to develop an equational connection calculus we take the Herbrand theorem as start point. This formulation is a consequence of the Herbrand theorem proved in [Pet92] for arbitrary quantifier free theories and formulas in non-clausal form.

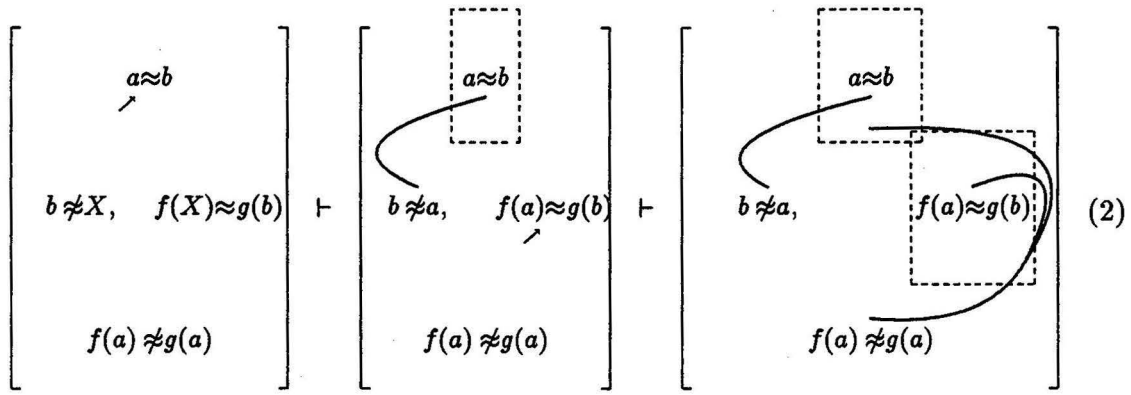
Theorem 3.1 *(***Herbrand theorem***)* *Any matrix M is $\mathcal{E}$-unsatisfiable if and only if there exist a set M' of variants of clauses of M, a substitution σ and a mating U which is spanning in M' such that for each $u \in U$ the instantiated eq-connection $\sigma(u)$ is $\mathcal{E}$-complementary.*

The pool calculus presented below is one of the algorithmic realizations of the criterion formulated in the Herbrand theorem. In order to explain it let us consider the following sample matrix. Clauses are displayed as rows. The arcs indicate the two elements of a mating which is spanning in the original matrix. The connections of this mating become $\mathcal{E}$-unsatisfiable if the substitution $\{X \mapsto a\}$ is applied to the matrix.



(1)

The existence of the spanning mating and of the unifier may be proved in a two step derivation. This derivation is displayed below by a sequence of structured matrices. Each of these matrices represents a set of paths which have to be examined yet. In the initial matrix (on the left) there have to be considered all paths which continue the empty path via the literal which is pointed by the arrow i.e. all paths. The pointed literal will be called the *actual goal.* In the second matrix the first connection has been found. It remains to consider those paths which continue the path $a \approx b$ (dashed boxed) via the literal $f(a) \approx f(b)$ (pointed by the arrow). In the matrix on the right nothing remains to be done. All continuations of the actual path (dashed boxed literals) contain the second connection. Consequently no actual goal is indicated.



(2)

The crucial data structure of the pool calculus that maintains the information which paths remain to be examined is a *pool* of so called *hooks*. Each hook is a pair (p, Γ) where p is a partial path in the matrix and Γ is a partial clause in the matrix such that $p \cap \Gamma = \emptyset$. The hook (p, Γ) will be denoted by $(p \perp \Gamma)$. The hook represents the paths which continue p via some element of Γ. p will be called the *actual path* of the hook and the elements of Γ are called the goals of the hook. Any hook of the form $(p \perp \emptyset)$ will be called a *solved hook*. Any hook of the form $(\emptyset \perp \Gamma)$ for some clause copy Γ is called an *initial hook*. Below we display the same derivation as before as a sequence of pools. Each pool represents the same sets of paths like the corresponding matrix from above.

$$\left\{ (\perp\ a{\approx}b) \right\} \vdash \left\{ \begin{array}{c} (\perp), \\ (a{\approx}b\ \perp\ f(a){\approx}g(b)) \end{array} \right\} \vdash \left\{ \begin{array}{c} (\perp), \\ (a{\approx}b\ \perp), \\ (a{\approx}b,\ f(a){\approx}g(b)\ \perp) \end{array} \right\} \quad (3)$$

In any inference step a hook called the *chosen hook* is deleted from the pool and some new hooks are produced and adjoined to the pool[2]. An *inference rule* is a quatruplet which is denoted in the form $\dfrac{h \quad \Gamma}{H}\ \sigma$ where (1) h is the chosen hook, (2) Γ is a (possibly empty) set of new copies of clauses from M, called the *extension clauses*, (3) H is a set of hooks, its elements are called *new hooks*, (4) σ is a substitution. The substitution determined by the inference rule has to be applied to the resulting pool.

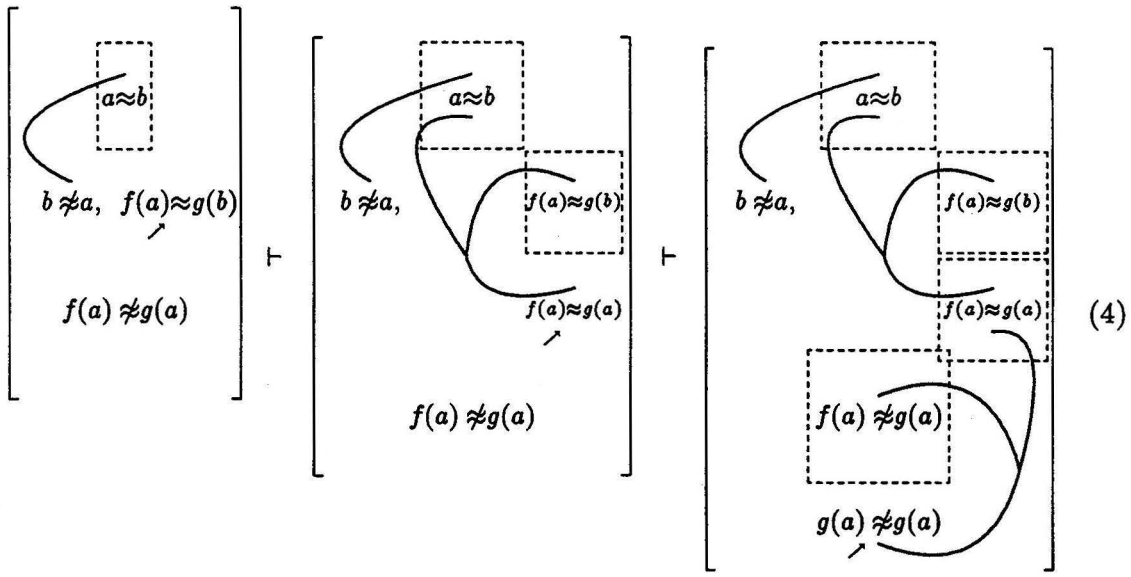
4 A partial eq-connection calculus

4.1 Basic notions for partial connection calculi

The idea of a partial eq-connection calculus is to support the approximative construction of both eq-connections and eq-unifiers by a number of steps which are of low complexity. This will be illustrated by the second inference step of the derivation (2). That single step may be broken into several substeps. In the first substep the goal $f(a){\approx}g(b)$ will be transformed into a new goal $f(a){\approx}g(a)$ by use of the equation $a{\approx}b$. This equation belongs to the eq-connection which has to be found in the considered inference step. In the second

[2]In an implementation the solved hooks will not be kept in the pool.

substep the transformed goal will be used in order to paramodulate into the inequality $f(a) \not\approx g(a)$. The resulting inequality $g(a) \not\approx g(a)$ is a complementary eq-connection.



(4)

It should be noticed that now there appear new goals like $f(a) \approx g(b)$ which do not occur in the original matrix. They will be treated by a second kind of hooks. We need the notions eq-residue and eq-hook for this purpose.

Definition 4.1 (Equational residue)

Suppose that K is not the empty path, Γ is a negative clause and Π is a unit clause or empty. Let q denote the path $K, \bar{\Gamma}, \bar{\Pi}$ (cf. section 2 for the notation). If σ is a substitution such that $\sigma(q)$ is a complementary eq-connection then the triplet $\langle \Gamma,\ \Pi,\ \sigma \rangle$ is called an *equational residue* or for short an *eq-residue* of the *key* K. The clauses Γ and Π will be called the *precondition* and the *paramodulant* for the key K. *(End definition)*

In the following table we present typical examples of pairs of keys and residues as they appear in well known calculi. They will be discussed later.

Example 4.1

Key	Residue			Remarks
	Preconditions	Paramodulant	Substitution	
$f(a) \approx b, f(X) \not\approx c$		$b \not\approx c$	$\{X \mapsto a\}$	paramodulation from $f(a)$ into $f(X)$
$f(a) \approx b, f(d) \not\approx c$	$a \not\approx d, b \not\approx c$			RUE-resolution between $f(a) \approx b$ and $f(d) \not\approx c$
$f(a) \approx b, f(d) \not\approx c$	$a \not\approx d$	$b \not\approx c$		relaxed paramodulation from $f(a)$ into $f(d)$

Bibliographical remark: The notion of residue in propositional theory reasoning is due to M. Stickel [Sti85]. A first-order version of this notion has been introduced

by P. Baumgartner [Bau92]. This notion has been applied in a partial theory model elimination calculus. However, Baumgartner's notion has the restriction that only one literal appears as part of the residue. The instantiations of our partial eq-connection calculus need residues with non-unit literal sets.

The residue will be applied in the calculus as follows. Suppose that a literal K_0 is a goal of a hook with actual path p. Suppose the literals $K_0, K_1, \ldots, K_n$ form a key. Then the literals $K_1, \ldots, K_n$ will be looked for on the path p or in clauses which are used for extension. However, the key alone does not form an eq-complementary connection. The goal K_0 will not be solved completely. Rather it will be substituted by new goals which are the elements of Γ, Π. Now we introduce a second kind of hooks as the formal tool for treating those partially solved goals.

Definition 4.2 (eq-pools, eq-hooks)

An *eq-hook* for a matrix M is a triplet (p, q, Γ) where p is a partial path in a set of copies M' of M, q is a literal set considered as a path, Γ is a Horn clause with body Δ such that[3] (1) $\mathcal{E} \models \bigwedge p \rightarrow \bigvee \Gamma$ and (2) $\mathcal{E} \models \bigwedge p \rightarrow (\bigvee \Delta \vee \bigwedge q)$. The eq-hook (p, q, Γ) will be denoted by $(p \parallel q \perp \Gamma)$. The eq-hook $(p \parallel q \perp \Gamma)$ represents all paths which continue the path p. An eq-hook of the form $(p \parallel q \perp \emptyset)$ will be called a *solved hook.* An *eq-pool* is a set of hooks and eq-hooks. *(End definition)*

4.2 The inference rules

In the present subsection we introduce the inference rules of a pool calculus which has been adapted to partial theory reasoning. Illustrating examples will be given in the next section 5.

Rule 1: Equality reduction

$$\frac{(p \perp u \not\approx v, \Delta)}{(p \perp \Delta)} \quad mgu(u, v)$$

Rule 2: Equality reduction within an eq-hook

$$\frac{(p \parallel q \perp u \not\approx v, \Delta)}{(p \parallel q \perp \Delta)} \quad mgu(u, v)$$

Remark: In the following rules the triplet $\langle \Gamma, \Pi, \sigma \rangle$ is an eq-residue for a key of the form K, L where K is a possibly empty path and L is a literal. We consider K as a path instead as a literal in order to cover different instantiations of the calculus by a notation which is as compact as possible.

Rule 3: Partial eq-reduction within a hook

$$\frac{(p \perp L, \Delta)}{(p, L \parallel \perp \Gamma, \Pi), (p \perp \Delta)} \quad \sigma$$

where $K \subseteq p$.

[3] $\bigwedge p$ (resp. $\bigvee \Gamma$) denotes the conjunction (resp. disjunction) of the elements of p (resp. of Γ.

Rule 4: Partial eq-reduction within an eq-hook

$$\frac{(p \parallel q \perp L, \Delta)}{(p \parallel q', q \perp \Gamma, \Pi, \Delta)} \quad \sigma$$

where $K \subseteq p \cup q$, $q' = \begin{cases} L & \ldots \text{ if } L \text{ is a positive equality literal} \\ \emptyset & \ldots \text{ otherwise} \end{cases}$.

Remark: K consists of a unique literal in the extension rules below.

Rule 5: Partial eq-extension from a hook

$$\frac{(p \perp L, \Delta) \qquad K, \Lambda}{(p, L, K \parallel \perp \Gamma, \Pi), \quad (p \perp \Delta), \quad (p, L \perp \Lambda)} \quad \sigma$$

Rule 6: Partial eq-extension from an eq-hook

$$\frac{(p \parallel q \perp L, \Delta) \qquad K, \Lambda}{(p, K \parallel q', q \perp \Gamma, \Pi, \Delta), \quad (p \perp \Lambda)} \quad \sigma$$

where $q' = \begin{cases} L & \ldots \text{ if } L \text{ is a positive equality literal} \\ \emptyset & \ldots \text{ otherwise} \end{cases}$.

In order to illustrate the introduced notions and the inference rules we show now the derivation (4) as a sequence of eq-pools. The underlined hooks are the chosen hooks.

$$\left\{ \begin{array}{c} (\perp), \\ \underline{(a \approx b \perp f(a) \approx g(b))} \end{array} \right\} \vdash_{\text{Rule 3}} \left\{ \begin{array}{c} (\perp), \\ (a \approx b \perp), \\ \underline{\left(\begin{array}{c} a \approx b, \\ f(a) \approx g(b) \end{array} \parallel \perp f(a) \approx g(a) \right)} \end{array} \right\}$$

$$\vdash_{\text{Rule 6}} \left\{ \begin{array}{c} (\perp), \\ (a \approx b \perp), \\ \left(\begin{array}{c} a \approx b, \\ f(a) \approx g(b) \end{array} \perp \right), \\ \left(\begin{array}{c} a \approx b, \\ f(a) \approx g(b), \\ f(a) \not\approx g(a) \end{array} \parallel f(a) \approx g(a) \perp g(a) \not\approx g(a) \right) \end{array} \right\} \quad (5)$$

The remaining unsolved eq-hook may be solved by rule 2. The shown inference steps are sound because every path which is not represented in the resulting pool contains a complementary eq-connection.

Proposition 4.1 *The partial eq-connection calculus is sound.*

5 Instances of the partial equational calculus

In the present section we instantiate the eq-residue in different ways in order to obtain eq-connection calculi with paramodulation, RUE-resolution and relaxed paramodulation.

5.1 Paramodulation

Definition 5.1 (Paramodulation residue)

Suppose that literals $l \approx r$ and L are given. Moreover there is a non-variable address α in L and a substitution σ such that $\sigma = mgu(L/\alpha, l)$. Then the pair $\langle L[\alpha \leftarrow r],\ \sigma\rangle$ is called a *paramodulation residue* for the *key* $l \approx r, L$. *(End definition)*

If $\langle L[\alpha \leftarrow r],\ \sigma\rangle$ is a paramulation residue then $\sigma(l \doteq r),\ \sigma(L),\ \sigma(\bar{L})[\alpha \leftarrow \sigma(r)]$ is a complementary eq-connection. Therefore $\langle \emptyset,\ L[\alpha \leftarrow r],\ \sigma\rangle$ is an eq-residue for the key $l \doteq r, L$.

Example 5.1 We continue the discussion of the derivation (5) in subsection 4.1. Let us consider a paramodulation derivation which detects the second eq-connection in the mating 1. The derivation consists of the inferences 1.-3.

1. Partial eq-reduction: paramadulation from b within an equation on the path into b within the unique goal of the hook:

$$\frac{(a \approx b \ \bot\ f(a) \approx g(b))}{(a \approx b, f(a) \approx g(b) \ \|\ \bot\ f(a) \approx g(a)),\ (a \approx b \ \bot\ )} \quad \{\,\}$$

2. Partial eq-extension: paramadulation from the term $f(a)$ within the goal literal of the chosen eq-hook into the term $f(a)$ within clause 3:

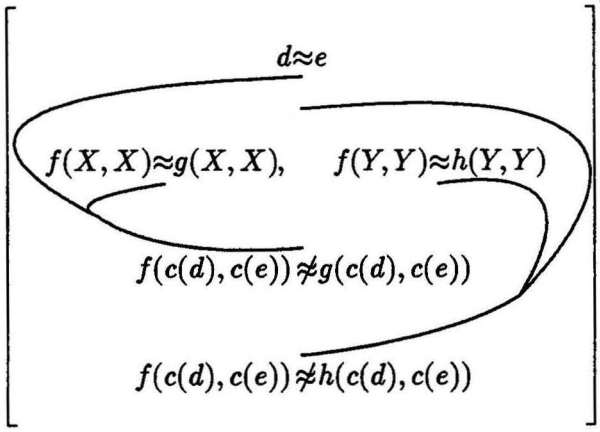
$$\frac{(a \approx b, f(a) \approx g(b) \ \|\ \bot\ f(a) \approx g(a)) \qquad f(a) \not\approx g(a)}{(a \approx b, f(a) \approx g(b), f(a) \not\approx g(a) \ \|\ f(a) \approx g(a) \ \bot\ g(a) \not\approx g(a)),\ (a \approx b, f(a) \approx g(b) \ \bot\ )} \quad \{\,\}$$

3. Equality reduction within an eq-hook:

$$\frac{(a \approx b, f(a) \approx g(b), f(a) \not\approx g(a) \ \|\ f(a) \approx g(a) \ \bot\ g(a) \not\approx g(a))}{(a \approx b, f(a) \approx g(b), f(a) \not\approx g(a) \ \|\ f(a) \approx g(a) \ \bot\ )} \quad \{\,\}$$

Example 5.2 (Counterexample contradicting the completeness of the pool calculus with paramodulation)

It may be seen easily that the following matrix (6) is $\mathcal{E}$-unsatisfiable. But there does not exist a refutation within the introduced calculus. The reason is that in any way there will appear a hook where either $f(X,X) \approx g(X,X)$ or $f(Y,Y) \approx h(Y,Y)$ remains as subgoal to be solved. But there is no possibility for a rule application. The partial eq-extension or partial eq-reduction are impossible because none of the sides of those equations may be unified with terms occurring in $d \approx e$, $f(c(d), c(e)) \not\approx g(c(d), c(e))$ or $f(c(d), c(e)) \not\approx h(c(d), c(e))$. Also none of the sides of $d \approx e$ may be unified with a non-variable occurrence in $f(X,X) \approx g(X,X)$ or $f(Y,Y) \approx h(Y,Y)$. Ofcourse, the equality reduction step is impossible too.



(6)

However, a paramodulation, say from $f(Y,Y)$ into $f(c(d),c(e))$ would be possible if the requirement of the paramodulation rule would be relaxed. This idea will be used by the relaxed paramodulation calculus which will be discussed in subsection 5.3.

5.2 RUE-resolution like calculus

The following notion is a simplified version of a notion introduced by V. Digricoli [DH86]. This notion prepares the definition of the RUE-residue which will be considered in this subsection as our next instantiation of the eq-residue.

Definition 5.2 (disagreement set)

(1) The unit clause $s \not\approx t$ is a disagreement set of the pair of terms $\langle s,t\rangle$.

(2) If for $i, i = 1,\ldots,n$ D_i is a disagreement set of pair $\langle s_i,t_i\rangle$ then the clause $D_1,\ldots,D_n$ is a disagreement set of the set of term pairs $\{\langle s_1,t_1\rangle,\ldots,\langle s_n,t_n\rangle\}$.

(3) For every n and n-ary function symbol f if for $i, i = 1,\ldots,n$ D_i is a disagreement set of the pair $\langle s_i,t_i\rangle$ then the clause $D = D_1,\ldots,D_n$ is a disagreement set of the term pair $\langle s,t\rangle$ where $s = f(s_1,\ldots,s_n)$ and $t = f(t_1,\ldots,t_n)$. Moreover, we say that D is below of $D' = \{s \not\approx t\}$, in symbols $D' \succ D$. By some abuse of notation we will denote the transitive closure of $\succ$ by the same symbol..

(4) A disagreement set D of a set of term pairs S is called *topmost* (resp. *downmost*) iff there is no disagreement set D' of S such that $D' \succ D$ (resp. $D \succ D'$).

(5) Any disagreement set of the term pair $\langle s,t\rangle$ is a disagreement set of the eq-literal $s \not\approx t$. Any disagreement set of the set of term pairs $\{\langle t,r\rangle,\langle s,l\rangle\}$ is a disagreement set of the eq-connection $l \approx r, s \not\approx t$.

(End definition)

Definition 5.3 (RUE-residue)

Let K be an eq-connection which consists of one or two elements. Let the clause $D = D' \dot{\cup} D''$ be a disagreement set for K. Moreover, let σ be $mgu(D'')$. Then the pair $\langle D',\ \sigma\rangle$ is called a *RUE-residue* for the *key* K. *(End definition)*

If $\langle D,\ \sigma\rangle$ is a RUE-residue for the key K and if p denotes the path $\bar{D}$ then $\sigma(K),\ \sigma(p)$ is a complementary eq-connection. Thus the RUE-residue $\langle D,\ \emptyset,\ \sigma\rangle$ is an instance of the eq-residue for the key K.

Example 5.3 (Example derivation)

Let us reconsider the matrix 6 which served in example 5.2 as an illustration for the incompleteness of the extension procedure with the paramodulation rule.

Because of lack of space let us use the following abbreviations for syntactic items:

Item	substituted by
$c(d)$	s
$c(e)$	t
$f(c(d),c(e)) \not\approx g(c(d),c(e))$	L
$f(X,X) \approx g(X,X)$	K
$f(Y,Y) \approx h(Y,Y)$	M

1. Partial eq-extension: The RUE-residue for the key K, L has been constructed from the disagreement set $X \not\approx s, X \not\approx t$ for that key keeping its second element und unifying its first one:

$$\frac{(\ \bot\ K, M) \qquad L}{(K, L \parallel\ \bot\ X \not\approx t),\quad (\ \bot\ M),\quad (K\ \bot\ )} \quad \{X \mapsto c(d)\}$$

2. Partial eq-reduction: The RUE-residue for the key $s \not\approx t$ consists of the disagreement set $d \not\approx e$ and the empty substitution:

$$\frac{(f(s,s) \approx g(s,s), L \parallel\ \bot\ s \not\approx t)}{(f(s,s) \approx g(s,s), L \parallel\ \bot\ d \not\approx e)} \quad \{\ \}$$

3. Partial eq-extension: The disagreement set for the key $d \not\approx e, d \approx e$ consists of the inequations $d \not\approx d, e \not\approx e$ which are both unified by the empty substitution.

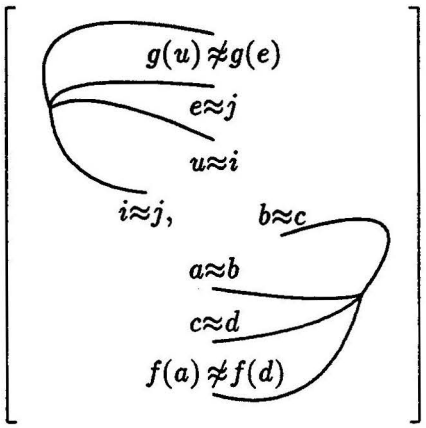
$$\frac{(f(s,s) \approx g(s,s), L \parallel\ \bot\ d \not\approx e) \qquad d \approx e}{(f(s,s) \approx g(s,s), L, d \not\approx e \parallel\ \bot\ ),\quad (f(s,s) \approx g(s,s), L\ \bot\ )} \quad \{\ \}$$

After step 3. there remains the hook $(\ \bot\ M)$ in the pool. The goal within this hook may be solved in a similar way like K. The remaining produced hooks have empty goal clauses. Thus, there may be constructed a successful derivation.

The RUE-residue solves the problem which appeared in the paramodulation derivation. There it was not possible to apply an inference rule to the goal $f(X,X) = g(X,X)$ because no unification of this literal or its subterms are possible which leads to a successful derivation. With the RUE-format it is possible to formulate this unification problem within the calculus and to solve it by the application of other equations. A negative goal is needed when the RUE-residue has to be constructed. Unfortunately, sometimes it is necessary to treat with positive goals. For this reason the extension procedure with the RUE-residue alone is incomplete. This will be shown in the following example.

Example 5.4 (Counterexample contradicting the completeness)

The following clause set is $\mathcal{E}$-unsatisfiable but there is no successful derivation of the pool calculus with the RUE-rule.



(7)

The counterexample has been constructed like the counter example for the paramodulation. In every derivation there appears $i \approx j$ or $b \approx c$ as subgoal to be solved. Again the rule is not applicable. However, it would be helpful if it would be possible to paramodulate. This gives raise to the last calculus which might be seen as a crossing between the paramodulation and the RUE-calculus.

5.3 Relaxed paramodulation

The counterexamples 5.2 in 5.1 and 5.4 in 5.2 show that the extension calculus is incomplete if the eq-residue is instantiated to the paramodulation residue or to the RUE-residue. The example 5.3 suggests that the addition of some capabilities of the RUE-residue might overcome the incompleteness of the extension procedure with the paramodulation residue.

In this subsection we instantiate the eq-residue to the relaxed paramodulation residue. In [SL91] W. Snyder and C. Lynch conjecture that set-of-support-resolution is complete with relaxed paramodulation. Unfortunately, they don't have a rigorous completeness proof. Also for the calculus below we don't have a completeness proof. Nevertheless, although we invested some effort into the search for counter examples we couldn't find them.

Definition 5.4 (Top unify)

Terms s and t *top unify* iff for each element of the downmost disagreement set for s and t holds that either both sides are equal or at least one side is a variable. *(End definition)*

Definition 5.5 (Relaxed paramodulation residue)

Suppose that literals $l \doteq r$ and L are given. Moreover, there is an address α in L such that L/α and l top unify. Let the clause $D = D' \dot{\cup} D''$ be the downmost disagreement set for L/α and l. Finally, let σ be $mgu(D'')$.

Then the pair $\langle D', L[\alpha \leftarrow r], \sigma \rangle$ is called a *relaxed paramodulation residue* for the *key* $l \doteq r, L$. *(End definition)*

If $\langle D', L[\alpha \leftarrow r], \sigma\rangle$ is a relaxed paramodulation residue and if p is the path $\bar{D}$ then

$$\sigma(l \doteq r),\ \sigma(L),\ \sigma(\bar{L})[\alpha \leftarrow \sigma(r)],\ \sigma(p)$$

is a complementary eq-connection. Thus, $\langle D,\ L[\alpha \leftarrow r],\ \sigma\rangle$ is an eq-residue for the key $l \doteq r, L$.

Example 5.5 We reconsider again the matrix 6 which served in example 5.2 as an illustration for the incompleteness of the extension procedure with the paramodulation rule. We use the same abbreviations as in example 5.3.

1. Partial eq-extension: We try relaxed paramodulation from $f(X,X)$ in the first literal of the second clause into $f(c(d),c(e))$ in the third clause. The residue for the key K, L has been constructed from the precondition $X \not\approx s, X \not\approx t$ for that key and the paramodulant $g(X,X) \not\approx g(c(d),c(e))$ unifying the first element of the precondition and keeping the remaining elements:

$$\frac{(\ \bot\ K, M) \qquad L}{(K, L \parallel\ \bot\ X \not\approx c(e), g(X,X) \not\approx g(c(d),c(e)))\quad (\ \bot\ M),\quad (K\ \bot\ )} \quad \{X \mapsto c(d)\}$$

2. and 3. Partial eq-extension followed by equality reduction: We paramodulate from e in the first clause into e in the first goal literal of the eq-hook. The equality reduction will be skipped.

$$\frac{(f(s,s) \approx g(s,s), L \parallel\ \bot\ c(d) \not\approx c(e), g(c(d),c(d)) \not\approx g(c(d),c(e))) \qquad d \approx e}{(f(s,s) \approx g(s,s), L, d \approx e \parallel\ \bot\ c(d) \not\approx c(d), g(c(d),c(d)) \not\approx g(c(d),c(e)))} \quad \{\ \}$$

4. Partial eq-reduction: We paramodulate from e in the equation $d \approx e$ on the path into e in the single goal of the eq-hook.

$$\frac{(f(s,s) \approx g(s,s), L, d \approx e \parallel\ \bot\ g(c(d),c(d)) \approx, g(c(d),c(e)))}{(f(s,s) \approx g(s,s), L, d \approx e \parallel\ \bot\ g(c(d),c(d)) \not\approx, g(c(d),c(d)))} \quad \{\ \}$$

To this eq-hook and the hook $(K\ \bot\ )$ which has been produced in the first inference will be applied equality reduction (within an eq-hook). The remaining hook $(\ \bot\ M)$ may be solved in a similar way. Thus, a derivation with the eq-residue instantiated to the relaxed paramodulation residue may be constructed.

6 Towards an implementation

The pool calculus which has been introduced above has been implemented by the present author. In the implementation the eq-residue has been instantiated to the paramodulation residue and to order restricted paramodulation residue. Moreover, a variant with basic paramodulation and order restricted basic paramodulation has been implemented. The mentioned implementation are in fact interpreters of the pool calculus written in Quintus-Prolog. Tests showed that these interpreters suffer from a rather high overhead. Better performance shows a PTTP-like prover (cf. [Sti]) due to G. Neugebauer, TH Darmstadt, which has been enhanced by paramodulation. Actually we are going to implement in that environment also the more sophisticated instantiations of eq-residue which has been discussed in the present paper.

7 Conclusion

We introduced a framework which allows the integration of different strategies of equality reasoning into the connection method. The formal tool for the calculus is the pool calculus which has been extended by a second kind of hooks in order to treat partial theory reasoning. The work reported in this paper aims in the construction of an experimental tool for equality reasoning. Thus, implementations are in work. We hope that experimental results will give intuitions for the clarification of theoretical problems. The most intriguing problem is which combinations of different restrictions of the inference rules give complete calculi and which combinations are not complete but adequate in an intuitive sense.

References

[Bau92] P. Baumgartner. Theory Model Elimination. In H. J. Ohlbach, editor, *Proc. GWAI 92*, 1992. MP-I-Inf.

[BFP92] P. Baumgartner, U. Furbach, and U. Petermann. A Unified Approach to Theory Reasoning. Forschungsbericht 15/92, University of Koblenz, 1992.

[BG92] Leo Bachmair and Harald Ganzinger. Basic paramodulation and superposition. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction: Saratoga Springs, 15-18 June 1992*, pages 462–476, Berlin, Juny 1992. Springer-Verlag. Lecture Notes in Artificial Intelligence.

[Bib82] W. Bibel. *Automated Theorem Proving.* Vieweg, 1st edition, 1982.

[DH86] Vincent J. Digricoli and Malcolm C. Harrison. Equality-Based binary Resolution. *Journal of the Association for Computing Machinery*, 1986.

[GNPS92] J. Gallier, P. Narendran, D. Plaisted, and W. Snyder. Theorem Proving with Equational Matings and Rigid E-unification. *J. of ACM*, 1992.

[Lov78] D. Loveland. *Automated Theorem Proving - A Logical Basis.* North Holland, 1978.

[Mor69] J. B. Morris. E-Resolution: An Extension of Resolution to include the Equality Relation. In *Proc. IJCAI*, pages 287–294, 1969.

[NS91] Gerd Neugebauer and Torsten Schaub. A pool-based connection calculus. Technical Report AIDA-91–02, FG Intellektik, Technisch Hochschule Darmstadt., 1991.

[Pet92] U. Petermann. How to build in an open theory into connection calculi. *J. on Computer and Artificial Intelligence*, 11(2):105–142, 1992.

[Rus91] Michael Rusinowitch. Theorem-proving with Resolution and Superposition. *Journal of Symbolic Computation*, 11:21–49, 1991.

[RW69] G. A. Robinson and L. Wos. Paramodulation and Theorem Proving in First Order Theories with Equality. In Meltzer and Mitchie, editors, *Machine Intelligence 4*. Edinburg University Press, 1969.

[SL91] W. Snyder and C. Lynch. Goal directed strategies for paramodulation. In R. Book, editor, *Rewriting Techniques and Applications*, Lecture Notes in Computer Science No. 488, pages 150 – 161, Berlin, 1991. Springer.

[Sti] M.E. Stickel. A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog. SRI International Research Report Technicl Note 464, Artificial Intelligence Center.

[Sti85] M.E. Stickel. Automated deduction by theory resolution. *J. of Automated Reasoning*, 4(1):333–356, 1985.

The Analytic Tableaux for Linear Miniscoped Horn-Like Temporal Logic

Regimantas PLIUŠKEVIČIUS
Institute of Mathematics and Informatics,
Akademijos 4, Vilnius 2600, Lithuania
email: logica@sedcs.mii2.lt

Abstract. A new type of tableaux system is proposed for the complete class (called miniscoped Horn-like) of the first order linear temporal logic. The described system contains some non-logical axioms indicating the saturation of a derivation process in this system. The proposed system is analytic because it possesses the subformula property.

Key words. Linear temporal logic, sequent and tableau systems.

Introduction

It is well known that the methods of automated deduction play an important role in various questions of artificial intelligence and computer science. Many fundamental works on automated deduction are based on the resolution method (due to Robinson, see e.g. [3]). Among the alternative methods of automated deductions in some works (see e.g. [2]) the semantic tableaux method (due to Smullyan) are advocated. Resolution method descends from the techniques developed by Herbrand, while the semantic tableaux method descends from the works of Gentzen and Beth. Gentzen's and Herbrand's works are closely related, but the resolution and semantic tableaux have different flavors to them. Tableaux systems have more flexible theorem provers and more intuitive and richer syntax than the systems based on resolution. Among tableaux-like calculi rather important are the so-called analytic ones, which are related to Gentzen-like calculi with the so-called subformula property.

It is well known that the temporal logics became a very important tool for solving various problems of computer science. The purpose of this paper is to present a new type of tableaux deduction, denoted by the TSat, for the complete class [5] (called miniscoped Horn-like) of the first order linear temporal logic with $\bigcirc$(next) and $\Box$(always). The object of consideration in this class is the formulas of the form $\bigwedge_{i=1}^{n}((A_i \supset)^{\circ}\Box^{\circ}B_i)$ $(n \geqslant 1)$, where A_i does not contain a positive occurrence of $\Box, (A_i \supset)^{\circ} \in \{\varnothing, A_i \supset\}, \Box^{\circ} \in \{\varnothing, \Box\}, B_i$ does not contain positive occurrence of $\Box$, besides, in their temporal parts quantifiers enter only the formulas of the form $Q\overline{x}E(Q \in \{\forall, \exists\})$, where E is an atomic formula. Such formulas will be

called MH-formulas. Despite of their simplicity, MH-formulas can express, for example, liveness properties of the form $\Box(A \supset \Diamond B)$. The class of MH-formulas is chosen for simplification of technical details and for clarity of the foundation of TSat. The method of constrution of TSat can be applied to a more general class of formulas and to other non-classical logics containing induction-like postulates. The deduction TSat directly corresponds to the sequential calculus Sat, introducted in the paper. (It conforms to the opinion (see e.g. [6]) that tableau systems are computer aided variant of the sequential calculi). TSat and Sat contain (instead the induction-like rules of inference) some non-logical axioms indicating the saturation of a derivation process in these calculi. These non-logical axioms can be regarded as a new deduction principle, named the "saturation" one, which replaces the induction-like postulates. For the justification of Sat (and, therefore, TSat) the Gentzen-type sequential infinitary calculus $G_{L\omega}$ is introduced. To clarify the role of the non-classical axioms of Sat (and TSat) the finitary Gentzen-type sequential calculus IN is introduced. The main rule of inference in IN is the so-called invariant rule having the form of some analytic cut-like rule of inference. The "cut formula" of this rule of inference is constructing from the non-logical axioms of Sat. The equivalence of the calculi $G_{L\omega}$, Sat, IN and TSat is proved for the MH formulas.

Let us note that all known tableau systems (see, e.g. [7])for temporal logics are not closely related to the sequential ones, they are not analytic, consider only decidable case of temporal logic, non-optimaly treat a positive occurrence of $\Box$ and do not include the saturation principle, which is the main feature of the system TSat. Besides, TSat is applicable both to decidable and undecidable case of linear temporal logic.

1. Description of the infinitary calculus $G_{L\omega}$

The formulas are determined with the help of logical symbols and temporal operator $\Box$ as usual. Instead of formulas of the type $\bigcirc A$ we shall consider the formulas with indices (denoted by A^i and certifying the truth value of A in the i-th moment of time) defined as follows: 1) $(E^i)^k := E^{i+k}$, where E is an atomic formula, i is zero (which is identified with an empty word) or any natural number, k is any natural number; 2) $(A \odot B)^k := A^k \odot B^k$, $\odot \in \{\supset, \vee, \wedge\}$; 3) $(\sigma A)^k := \sigma A^k$, $\sigma \in \{\neg, \Box, \forall x, \exists x\}$.

A sequent is an expression of the form $\Gamma \to \Delta$, where Γ, Δ are arbitrary finite sets (not sequences or multisets) of formulas.

The calculus $G_{L\omega}$ is defined by the following postulates.

Axiom: $A \to A$.

Rules of inference:

1) temporal ones:

$$\frac{A, \Box A^1, \Gamma \to \Delta}{\Box A, \Gamma \to \Delta}(\Box \to) \qquad \frac{\{\Gamma \to \Delta, A^k\}_{k \in \omega}}{\Gamma \to \Delta, \Box A}(\to \Box_\omega),$$

where $k \in \omega$ means that $k \in \{0, 1, \ldots\}$; here and below Γ^1 means $A_1^{k_1+1}, \ldots, A_n^{k_n+1}$, if $\Gamma = A_1^{k_1}, \ldots, A_n^{k_n}$, $n \geqslant 1$, $k_i \geqslant 0$, $1 \leqslant i \leqslant n$;

2) logical rules of inference consist of traditional invertible rules of inference;

3) structural rules: (W) (weakening); from the definition of the sequent it follows that $G_{L\omega}$ implicity contains the structural rules contraction and exchange.

Derivations in $G_{L\omega}$ are built in an usual way (for the calculi with ω-rule), i.e. in the form of an infinite tree (with finite branches); the height of a derivation D is an ordinal (defined in a traditional way). Let I be some calculus, then notation $I \vdash S$ means that the sequent S is derivable in S.

A derivation in $G_{L\omega}$ will be called atomic if all axioms have the form $E \to E$, where E is an atomic formula, i.e. $E = P^i(t_1, \ldots, t_n)$ $(i \geqslant 0)$.

Lemma 1.1. An arbitrary derivation in $G_{L\omega}$ may be transformed into an atomic one.

Proof: using the rules of inference of $G_{L\omega}$.

Remark 1.1 All derivations in $G_{L\omega}$ will be regarded as atomic ones.

Lemma 1.2 (invertibility of the rules of inference of $G_{L\omega}$). Let (i) be the rule of inference of $G_{L\omega}$, (except (W)), S be the conclusion of (i), S_1 be any premise of (i), then $G_{L\omega} \vdash S \Longrightarrow G_{L\omega} \vdash S_1$.

Proof: by induction on the height of the given derivation.

Theorem 1.1 (a). The calculus $G_{L\omega}$ is sound and complete; (b) the cut rule is admissible in $G_{L\omega}$.

Proof: see [4, 5].

2. Description and investigation of the sequential saturated calculus Sat

Let $S = A_1, \ldots, A_n \to B_1, \ldots, B_m$, then $S^F = \bigwedge\limits_{i=1}^{n} A_i \supset \bigvee\limits_{i=1}^{m} B_i$. The sequent S will be called MH-sequent, if S^F is equivalent to the MH-formula. At first, let us defined the canonical form of MH-sequents (simply: sequents). A sequent S will be called primary, if $S = \Sigma_1, \Pi_1^1, \Box\Omega^1 \to \Sigma_2, \Pi_2^1, \Box\Delta^1$, where $\Sigma_i = \varnothing$ $(i = 1, 2)$ or consists of logical formulas without indices; $\Pi_i^1 = \varnothing$ $(i = 1, 2)$ or consists of logical formulas with indices; $\Box\Omega^1 = \varnothing$ or consists of arbitrary miniscoped formulas of the form $\Box A^1$; $\Box\Delta^1 = \varnothing$ or $\Box\Delta^1 = \Box B^1$ (as follows from the definition of MH-sequents, Ω^1, B^1 does not contain positive (with respect to S) occurrences of $\Box$). A sequent S will be called quasiprimary, if $S = \Pi_1, \Box\Omega \to \Pi_2, \Box\Delta$, where $\Box \notin \Pi_1, \Pi_2$. It is clear that every primary sequent is the quasiprimary one. A sequent S will be called ordinary, if S contains both negative and positive occurrences of $\Box$. A sequent S will be called singular (simple) if S does not contain negative (positive, respectively) occurrences of $\Box$. An ordinary primary (quasiprimary) sequent will be called proper, if $S \neq \Box\Omega^1 \to \Box B^1$ $(S \neq \Box\Omega \to \Box B$, respectively).

The rules of inference of the calculus Sat consist of the rules of inference of $G_{L\omega}$, except $(\to \Box_\omega)$, and the following three rules of inference:

$$\frac{\Pi_1 \to \Pi_2, A;\ \Pi_1 \to \Pi_2, \Box A^1}{\Pi_1 \to \Pi_2, \Box A}(\to \Box^1) \quad \frac{\Box\Gamma \to A}{\Box\Gamma \to \Box A}(\Box) \quad \frac{S_i^*}{S^*}(A)\ (i \in \{1,2\}),$$

where $\Pi_j \neq \varnothing$ $(j \in \{1,2\})$ and $\Pi_1 \neq \Box\Gamma$; $\Box\Gamma = \Box B_1, \ldots, \Box B_m$ $(m \geqslant 0)$; S^* is a primary sequent, i.e. $S^* = \Sigma_1, \Pi_1^1, \Box\Omega^1 \to \Sigma_2, \Pi_2^1, \Box A^1$; $S_1^* = \Sigma_1 \to \Sigma_2$; $S_2^* = \Pi_1, \Box\Omega \to \Pi_2, \Box A$.

Lemma 2.1 (invertibility of the rules of inference of Sat). All rules of inference of Sat, except (W) and (A), are invertible in $G_{L\omega}$.

Proof: follows from invertibility of the rules of inference of $G_{L\omega}$ and admissibility of cut in $G_{L\omega}$.

Lemma 2.2 (disjunctional invertibility of (A)). Let S^* be the conclusion of (A), i.e. $S^* = \Sigma_1, \Pi_1^1, \Box\Omega^1 \to \Sigma_2, \Pi_2^1, \Box A^1$, then $G_{L\omega} \vdash S^* \Longrightarrow \mathrm{Log} \vdash \Sigma_1 \to \Sigma_2$ or $G_{L\omega} \vdash \Pi_1, \Box\Omega \to \Pi_2, \Box A$, where Log is the calculus obtained from $G_{L\omega}$ by dropping $(\to \Box_\omega), (\Box \to)$.

Proof: using induction on the height of the given derivation.

Remark 2.1. Instead of (A) one can apply a more effective (but slightly more complicatedly foundable) rule of inference (A^+), in which Σ_i $(i = 1,2)$ consists of logical formulas without indices and/or logical formulas with indices, not belonging to Ω^1; Π_i^1 $(i = 1,2)$ consists of logical formulas with indices belonging to Ω^1.

A derivation in Sat is constructed bottom-up applying the rules of inference of Sat. As follows from Lemmas 2.1, 2.2 all sequents from the derivation in Sat are derivable in $G_{L\omega}$. Some sequents which indicate the saturation of a derivation process in Sat play the role of non-logical axioms in Sat. To define the class of these sequents let us define the tactic of constructing a derivation in Sat. At first, let us define the notion of reduction (denoted by $R(S)\{i\} \Longrightarrow \{S_1, \ldots, S_n\}$, simply: $R(S)$) of a sequent S to the sequents $S_1, \ldots, S_n$, where $\{i\}$ is the set of rules of inference of Sat. $R(S)$ is a tree of sequents; the lowest sequent of $R(S)$ is the sequent S; every sequent in $R(S)$, except S, is the upper sequent of the rule of inference $(k) \in \{i\}$, whose lower sequent is also in $R(S)$, i.e. $R(S)$ consists of the bottom-up applications of the rule of inference from $\{i\}$. The leaves of $R(S)$ are the sequents $S_1, \ldots, S_n$ and any logical axiom.

Lemma 2.3. For every sequent S one can construct $R(S)\{i\} \Longrightarrow \{S_1, \ldots, S_n\}$, where $\forall i$ $(1 \leqslant i \leqslant)$ S_i is a primary (quasiprimary) sequent; $\{i\}$ is the set of rules of inference of Sat, except (A), $(\Box)$ and the rules of inference for quantifiers (and $(\Box \to), (\to \Box^1)$, respectively), besides $G_{L\omega} \vdash S \Longrightarrow G_{L\omega} \vdash S_i$ $(i = 1, \ldots, n)$.

Proof: follows from Lemma 2.1.

The set of the primary (quasiprimary) sequents from Lemma 2.3 will be denoted by $P(S)$ ($QP(S)$ respectively). Let I_1 be the calculus obtained from $G_{L\omega}$ by dropping $(\to \Box_\omega)$; I_2 (I_3) be the calculus, obtained from I_1 by adding $(\Box)$ ($(\Box)$ and (A), respectively).

Lemma 2.4. Let $G_{L\omega} \vdash S$, then (a) $I_1 \vdash S$, if S is simple; (b) $I_2 \vdash S$, if $S = \Box\Gamma \to \Box A$; (c) $I_3 \vdash S$, if S is singular.

Proof: follows from Lemmas 2.1, 2.2.

Let S be a proper quasiprimary sequent, i.e. $S = \Pi, \Box\Omega \to \Delta, \Box A$ (where $\Pi, \Delta \neq \varnothing$), let us define the notion of resolvent of the sequent S, denoted by $Re(S)$. Let us construct $P(S)$ and let $S_j^* \in P(S)$ $(j = 1, \ldots, n)$; let us bottom-up apply (A) to S_j^* and let S_{j2}^* be the "temporal" conclusion of this bottom-up application of (A) (i.e. $S_{j2}^* = \varnothing$, if $i = 1$ in (A)), then $Re(S) = \{S_{12}^*, \ldots, S_{n2}^*\} = \{S_1, \ldots, S_k\}$ $(k \leqslant n)$. We say that a sequent S absorbs the sequent S' (in symbols: $S' \prec S$ or $S \succ S'$), if can S' be obtained from S with the help of the structural rule of inference (W). Let us introduce the notion of the k-th resolvent of S, denoted by $Re^k(S)$ $(k \in \omega)$. $Re^k(S)$ is constructed as a tree of sequents, defined as follows $Re^0(S) = S$; $Re^1(S) = Re(S)$. Let $S_k \in Re^k(S)$, $S_l \in Re^l(S)$ $(k < l)$ and S_k, S_l belong to the same branch; we say that S_k is saturated, if $S \succcurlyeq S_l$ ($S' \preccurlyeq S''$ means $S' = S''$ or $S' \prec S''$); the sequent S_l is called absorbed one. Let $S_k \in Re^k(S)$, we say that the sequent S_k is blocked if S_k is absorbed and $\forall j$ $(f \leqslant j < k-1)$ $S_j \in Re^j(S)$ is saturated; f is some natural number and S_f, $S_{f+1}, \ldots, S_k$ belong to the same branch; otherwise $S_k \in Re^k(S)$ will be called non-blocked. Let $S_i \in Re(S)$ and S_i be non-blocked, then $Re^{k+1}(S) = \bigcup\limits_i Re(S_i)$. We say that a derivation D in Sat is constructed using the resolvent tactic, if D is constructed by generating $Re^k(S)$ $(k \in \omega)$. A branch B in the derivation D will be called saturated if the leaf of B is blocked. A branch B in the derivation D in Sat will be called closed if the leaf of B is either a logical axiom or B is saturated. The derivation D in Sat of the sequent S will be called closed (in symbols: Sat $\vdash^D S$) if all branches of D are closed. Let Sat $\vdash^D S$, then the set of saturated sequents from D will be denoted by Sat $\{S\}$. It is easy to see that the same sequent may have several sets Sat $\{S\}$.

Let us describe the tactic of constructing the derivation of an arbitrary MH-sequent S in Sat. Let us construct $QP(S) = \{S_1, \ldots, S_n\}$. If S_i is either simple or singular, or $S_i = \Box\Gamma \to \Box A$, then let us try to derive S_i in I_j $(j = 1, 2, 3)$ (see Lemma 2.4). If S_i is proper quasiprimary, then let us try to derive S_i by generating $Re^k(S_i)$.

Example 2.1. (a) Let $S = P(c)$, $\Box\Omega_1$, $\Box\Omega_2, \Box\Omega_3 \to \Box Q(c)$, where $\Omega_1 = (\exists x P(x) \supset \forall x P^1(f(x)))$, $\Omega_2 = (\exists x P(x) \supset \forall x Q(x))$, $\Omega_3 = (\exists x Q(f(x)) \supset \forall x Q(x))$. Then $P(S) = P(c)$, $\forall x P^1(f(x))$, $\Box\Omega_1^1$, $\Box\Omega_2^1$, $\Box\Omega_3^1$, Ω_2, $\Omega_3 \to \Box Q^1(c)$. Let us generate $Re^k(S)$: $Re^\circ(S) = S$; $Re^1(S) = Re(S) = S^* = \forall x P(f(x)), \Box\Omega_1, \Box\Omega_2, \Box\Omega_3, \to \Box Q(c) = Re^2(S)$. Therefore Sat $\vdash S$ and Sat $\{S\} = S^*$. It is easy to verify that $G_{L\omega} \vdash Re^k(S), k \in \{0, 1\}$;
(b) Let $S = \forall x P(x), \Box\Omega \to \Box P(f(b))$, where $\Omega = \exists x P(x) \supset \forall x P^1(f(x))$, then it is easy to verify that Sat $\vdash^D S$, where D contains one saturated branch and Sat $\{S\} = \forall x P(f(x)), \Box\Omega \to \Box P(f(b))$;
(c) Let $S = P, Q, \Box\Omega \to \Box Q$, where $\Box\Omega = \Box(P \supset P^2), \Box(P^2 \supset Q^2), \Box(Q^2 \supset Q^1)$. Then it is easy to verify that Sat $\vdash^D S$ and D contains two saturated branches:

$\mathrm{Sat}\{S\} = \{S_1, S_2\}$, where $S_1 = P, Q, \Box\Omega \to \Box Q$; $S_2 = P^1, Q^1, Q, \Box\Omega \to \Box Q$. In one branch $S_1 \succ P, Q, (P^1 \supset Q^1), (Q^1 \supset Q), \Box\Omega \to \Box Q = S_1'$ and $S_2 \succ P^1, Q^1, Q, (P \supset Q), \Box\Omega \to \Box Q = S_2'$. In another branch $S_1 \succ P, Q, P^1, (P^1 \supset Q^1), (Q^1 \supset Q), \Box\Omega \to \Box Q = S_3'$. If we do not notice that $S_i \succ S_i'$ $(i = 1, 2)$, then we get another $\mathrm{Sat}\{S\} = \{S_1' S_2', S_3', S_4'\}$, where $S_4' = P, P^1, Q^1, Q, \Box\Omega \to \Box Q$.
(d) Let $S = A, P, \Box\Omega \to \Box A$, where $\Box\Omega = \Box(P \supset (A \wedge Q)), \Box(Q \supset (A^1 \wedge P^1))$ (see [1]), then it is easy to verify that $\mathrm{Sat} \vdash^D S$ and $\mathrm{Sat}\{S\} = A, P, \Box\Omega \to \Box A$.

Remark 2.2. Let us consider the "prefixed" variant of the sequent S from Example 2.1(a), i.e. let $S = P(c), \Box\forall x[(P(x) \supset P^1(f(x))) \wedge (P(x) \supset Q(x)) \wedge (Q(f(x)) \supset Q(x))] \to \Box Q(c)$, then it is easy to verify that $G_{L\omega} \vdash S$, but $\mathrm{Sat} \nvdash S$.

Let us define the notion of subformula and resolvent subformula of an arbitrary formula of a MH-sequent. These notions are obtained from the notion of subformula of an arbitrary first order formula by adding the following points: (1) the subformulas (resolvent subformulas) of A^1 are the subformulas (resolvent subformulas) of A; (2) subformulas (resolvent subformulas of $\Box B$ are subformulas of B and $\Box B^1$ (resolvent subformulas of B); (3) subformulas (resolvent subformulas) of the quasiatomic formula $Q\bar{x}E(\bar{x})$ are the subformulas of $E(\bar{t})$ (where $\bar{t} = t_1, \ldots, t_n$), t_i is an arbitrary term) (and it is the formula $Q\bar{x}E(\bar{x})$ itself). Therefore, the set of subformulas (resolvent subformulas) of an arbitrary formula, containing $\Box$, is infinite, even in a propositional case (finite, respectively, even in non-propositional case). It is obvious that the calculus Sat possesses subformula property, besides if $\mathrm{Sat} \vdash^D S$, where S is a primary sequent then all sequent from D which are tracetable to saturated sequents consist of a resolvent subformulas of formulas from S. Let Sat' be the calculus obtained from Sat by dropping $(\Box)$.

Lemma 2.5. Let S be any proper ordinary quasiprimary sequent and let $\mathrm{Sat}' \vdash^D S$, $S_i \in \mathrm{Sat}\{S\}$ $(i = 1, \ldots, n)$. Then $\forall i$ $(1 \leqslant i \leqslant n)$ $S_i = \Pi_i, \Box\Omega \to \Delta_i, \Box A$, where Π_i, Δ_i consists of resolvent subformulas of formulas from Ω, or $S_i = \Box\Omega \to \Box A$; besides $G_{L\omega} \vdash S \Longrightarrow G_{L\omega} \vdash S_i$ $(i = 1, \ldots, n)$ (a).

Proof. The form of S_i follows from the construction of D; the point (a) follows from Lemmas 2.1, 2.2.

Lemma 2.6. Let $\mathrm{Sat}' \vdash S$ and $S' \in \mathrm{Sat}\{S\}$, then $Re(S') = \{S_1, \ldots, S_n\}$, where $\forall i$ $(1 \leqslant i \leqslant n)$ $S_i \preccurlyeq S^* \in \mathrm{Sat}\{S\}$.

Proof: follows from definitions of $Re(S)$ and $\mathrm{Sat}\{S\}$.

Example 2.2. Let S be the sequent from Example 2.1(c), and let $\mathrm{Sat}\{S\} = \{S_1, S_2\}$, then $Re\{S_1\} = \{S_2\}$, $Re\{S_2\} = \{S_1', S_2'\}$, $S_1' \prec S_1$, $S_2' \prec S_2$. Let $\mathrm{Sat}\{S\} = \{S_1', S_2', S_3', S_4',\}$, then $Re\{S_1'\} = S_2'$; $Re\{S_2'\} = \{S_1', S_4'\}$; $Re\{S_3'\} = S_4'$; $Re\{S_4'\} = S_3'$.

3. Description of an invariant calculus IN and the equivalence of $G_{L\omega}$, Sat and IN for MH-sequents

Let us introduce the invariant calculus IN. The postulates of the calculus IN are

obtained from the calculus $G_{L\omega}$ replacing $(\to \Box_\omega)$ by $(\to \Box^1), (\Box)$ and the following one:

$$\frac{\Gamma, \Box\Omega \to \Delta, R; \quad R \to R^1; \quad R \to A}{\Gamma, \Box\Omega \to \Delta, \Box A} (\to \Box),$$

where (1) $\Gamma, \Box\Omega \to \Delta, \Box A \in \mathrm{Sat}\,\{S\} = \{\Pi_1, \Box\Omega \to \Delta_1, \Box A, \ldots, \Pi_n, \Box\Omega \to \Delta_n, \Box A\}$ and S is a proper ordinary primary sequent such that $\mathrm{Sat}' \vdash S$; (2) $R = \bigvee\limits_{i=1}^{n} (\Pi_i^\wedge \wedge \neg\Delta_i^\vee) \wedge \Box\Omega$ (where $\Gamma^\wedge(\Gamma^\vee)$ means conjunction (disjunction) of formulas from Γ); the formula R is called an invariant formula.

Remark 3.1. It is clear that the rule of inference $(\to \Box)$ destroys the subformula property (it becomes analytic cut-like rule of inference after adding the conditions (1), (2), see above), which is restored by means of saturation principle.

Lemma 3.1. In the calculi $G_{L\omega}$, IN the following rule of inference: $\frac{\Gamma \to \Delta}{\Gamma^1 \to \Delta^1}$ $(+1)$ is admissible.

Proof: by induction on the height of the derivation of the sequent $\Gamma \to \Delta$.

Example 3.1. Let S be the same as in Example 2.1(c) and let $\mathrm{Sat}\,\{S\} = \{S_1, S_2\}$, then the invariant formula $R = ((P \wedge Q) \vee (P^1 \wedge Q^1 \wedge Q)) \wedge \Box\Omega$. It is easy to verify that $\mathrm{Prop} \vdash P, Q, \Box\Omega \to R$ (1), where Prop is the propositional calculus); $I_1' \vdash R \to R^1$ (2), where I_1' is the calculus obtained from $G_{L\omega}$ by dropping $(\to \Box_\omega)$); $\mathrm{Prop} \vdash R \to A$ (3). Applying $(\to \Box)$ to (1), (2), (3) we get $IN \vdash S$.

Lemma 3.2. In the calculus $G_{L\omega}$ the rule of inference $(\to \Box)$ is admissible.

Proof: using admissibility of cut and Lemma 3.1.

Lemma 3.3. The rule of inference $(\to \Box^1)$ is admissible in $G_{L\omega}$.

Proof: using admissibility of cut in $G_{L\omega}$.

Lemma 3.4. The rule of inference $(\Box)$ is admissible in $G_{L\omega}$.

Proof: using Lemma 3.1 and $(\Box \to), (W), (\to \Box_\omega)$.

Theorem 3.1. $IN \vdash S \Longrightarrow G_{L\omega} \vdash S$.

Proof: follows from Lemmas 3.2, 3.3, 3.4.

Now we prove that $\mathrm{Sat} \vdash S \Longrightarrow G_{L\omega} \vdash S$. Let I be the calculus obtained from $G_{L\omega}$ replacing $(\to \Box_\omega)$ by $(\to \Box^1)$.

Lemma 3.5. Let $\mathrm{Sat} \vdash S$ and $S_i = \Pi_i, \Box\Omega \to \Delta_i, \Box A \in \mathrm{Sat}\,\{S\}$ $(i = 1, \ldots, n)$, then $\forall i$ $(1 \leqslant i \leqslant n)$ $I \vdash \Pi_i, \Box\Omega \to \Delta_i, B \Longrightarrow I \vdash \Pi_i, \Box\Omega \to \Delta_i, B^1$.

Proof. As follows from Lemma 2.6 $\forall i$ there exists $Re(S_i) = \{S_{i_1}, \ldots, S_{i_m}\}$ such that $\forall j$ $(1 \leqslant j \leqslant m)$ $S_{i_j} \in \mathrm{Sat}\,\{S\}$ or $S_{i_j} \prec S' \in \mathrm{Sat}\,\{S\}$. Let us consider $R(S_i)\{k\} \Longrightarrow \{S_{i_1}, \ldots, S_{i_m}\}$, i.e. reduction of S_i to the sequents $S_{i_1}, \ldots, S_{i_m}$. As follows from the definition of $Re(S_i)$ we get that $\{k\} \in \{\text{logical rules of inference}\} \cup \{(\to \Box^1), (A)\}$. In each branch of this reduction there is only one bottom-up ap-

plication of (A) :

$$\frac{S^* = \Pi_j^*, \Box\Omega \to \Delta_j^*, \Box A}{\Sigma_1, \Pi_j^{*1}, \Box\Omega \to \Sigma_2, \Delta_j^{*1}, \Box A^1}\,(A)$$

$$\vdots$$

$$\Pi_i, \Box\Omega \to \Delta_i, \Box A,$$

where $\Pi_j^* = \Pi_j$; $\Delta_j^* = \Delta_j$, if $S^* \in \mathrm{Sat}\,\{S\}$ and $\Pi_j^* = \Pi_j', \Pi_j$; $\Delta_j^* = \Delta_j', \Delta_j$, if $S^* \prec S' \in \mathrm{Sat}\,\{S\}$. Let us perform in $R(S_i)$ (for all i) the following operations: (1) if $S^* \prec S' \in \mathrm{Sat}\,\{S\}$, then let us bottom-up apply (W) to S^* (with the sequent S' as the premise of (W)); (2) above (A) let us replace $\Box A$ by B; (3) below (A) let us replace $\Box A^1$ by B^1. Then instead of S^* we get the sequent $\Pi_j^*, \Box\Omega \to \Delta_j^*, B$, which is derivable in I by assumption of the Lemma (and by (W), if $S^* \prec S' \in \mathrm{Sat}\,\{S\}$). Instead of bottom-up applications of (A) we get (W) and $(+1)$ (which is admissible in I). All bottom-up application of rule of inference (k), except (A), in the reduction $R(S_i)$ we get the application of the same rule of inference (k). Therefore, instead of the reduction $R(S_i)$ we get $I \vdash \Pi_i, \Box\Omega \to \Delta_i B^1$, for all i.

Theorem 3.2. $\mathrm{Sat} \vdash^D S \Longrightarrow G_{L\omega} \vdash S$.

Proof: by Lemmas 2.1, 2.3, 2.4 it is sufficient to consider the case when S is a proper ordinary quasiprimary sequent. Applying Lemma 3.5 and $(\to \Box_\omega)$ we get $G_{L\omega} \vdash S_i \in \mathrm{Sat}\,\{S\}$. All applications of (A) in D may be replaced by (W) and $(+1)$ (which is admissible in $G_{L\omega}$), all application of $(\to \Box^1)$ are admissible in $G_{L\omega}$ by Lemma 3.3, therefore $G_{L\omega} \vdash S$.

Now, we prove that $\mathrm{Sat} \vdash S \Longleftrightarrow IN \vdash S$.

Lemma 3.6. Let $\mathrm{Sat}' \vdash S$, then $I \vdash R \to R^1$, where R is the invariant formula, as indicated above.

Proof: using Lemma 3.5, taking R instead B.

Lemma 3.7. Let $\mathrm{Sat}' \vdash^D S$, then $\forall i\ (1 \leqslant i \leqslant n)\ IN \vdash S_i = \Pi_i, \Box\Omega \to \Delta_i, \Box A \in \mathrm{Sat}\,\{S\}$.

Proof. It is obvious that $\mathrm{Prop} \vdash \Pi_i, \Box\Omega \to \Delta_i, R$ (1). From Lemma 3.6 we get $I \vdash R \to R^1$ (2). From the construction of D it follows that $I_1' \vdash \Pi_i, \Box\Omega \to \Delta_i, A$ $(3_i')$ $(i = 1, \ldots, n)$ (where I_1' is the calculus obtained from $G_{L\omega}$ by dropping $(\to \Box_\omega)$). Applying $(\neg \to), (\wedge \to), (\vee \to)$ to $(3_i')$ we get $I_1' \vdash R \to A$ (3). Applying $(\to \Box)$ to (1), (2), (3) we get $IN \vdash S_i$.

Theorem 3.3. $\mathrm{Sat} \vdash S \Longleftrightarrow IN \vdash S$.

Proof. The part $\Longrightarrow$ follows from Lemma 3.7, the part $\Longleftarrow$ follows from the definition of IN.

Lemma 3.8 Let $S = \Pi, \Box\Omega \to \Delta, \Box A$ be a proper ordinary quasiprimary sequent, then $G_{L\omega} \vdash S \Longrightarrow \mathrm{Sat}' \vdash S$.

Proof. Starting from S let us construct a tree D of sequents, applying the resolvent tactic. It is follows from Lemmas 2.1, 2.2 and from the construction of D that we can reduce S either to logical axioms, or to the sequents $S_i = \Pi_i, \Box\Omega \to$

$\Delta_i, \Box A$, where Π_i, Δ_i consists of resolvent subformulas of the formulas from Ω. Since set of these subformulas is finite we get that all the leaves of $Re^k(S)$ (for some finite natural number k) are blocked; the leaves of other branches of D are logical axioms. Therefore $\mathrm{Sat}' \vdash^D S$.

Theorem 3.4. Let S be a MH-sequent, then $G_{L\omega} \vdash S \Longleftrightarrow \mathrm{Sat} \vdash S$.

Proof. The part $\Longrightarrow$ follows from Lemma 3.8, the part $\Longleftarrow$ follows from Theorem 3.2.

Theorem 3.5. Let S be a MH-sequent, then $G_{L\omega} \vdash S \Longleftrightarrow IN \vdash S$.

Proof: follows from Theorems 3.3, 3.4.

4. Construction of the saturated analytic tableaux

Let us describe the analytic tableaux system TSat for the saturated sequential calculus Sat. The derivation in TSat is nothing but a derivation in Sat turned upside-down and rewritten using other more effective notation. TSat is more computer-aided than Sat because it does not require to duplicate the extra formulae (see e.g. [3]) when constructing the derivations in TSat. The derivation in TSat represented a tree in the vertices of which there is only one formula. The tree is constructed from top to down starting from the initial vertex $P[A]$, where A is the given formula. In each subsequent vertex there is a formula of the form $P[B_1]$ or $N[B_2]$. These formulae correspond to the positive and negative parts of the initial formula A. To make TSat closer to Sat, we present TSat so that it (contrary to the traditional tableaux systems) construct the proof but not a refutation. The rules of inference for logical operators are obvious, therefore we shall indicate only rules of inference, corresponding $(\Box \rightarrow), (\rightarrow \Box^1), (\Box)$ and (A) (each rule of inference may be applied from any vertex of a branch of the tree):

$$(\Box^-) \quad \begin{array}{c} N[\Box A] \\ | \\ N[A] \\ | \\ N[\Box A^1] \end{array} \qquad \begin{array}{c} (\Box^1) P[\Box A] \\ / \quad \backslash \\ P[A] \quad P[\Box A^1] \end{array}$$

$$(A) \quad \begin{array}{c} N[\Sigma_1,\ \Pi_1^1,\ \Box\Omega^1], \quad P[\Sigma_2,\ \Pi_2^1,\ \Box A^1] \\ / \qquad\qquad\qquad \backslash \\ N[\Sigma_1], P[\Xi_2] \quad N[\Pi_1, \Box\Omega],\ P[\Pi_2, \Box A] \end{array}$$

$$(\Box) \quad \begin{array}{c} N[\Box\Gamma],\ P[\Box A] \\ | \\ N[\Box\Gamma],\ P[\Box A], \end{array}$$

where $\delta[\nabla] = \delta[A_1], \ldots, \delta[A_n]$ $(\delta \in \{P, N\})$, if $\nabla = A_1, \ldots, A_n$.

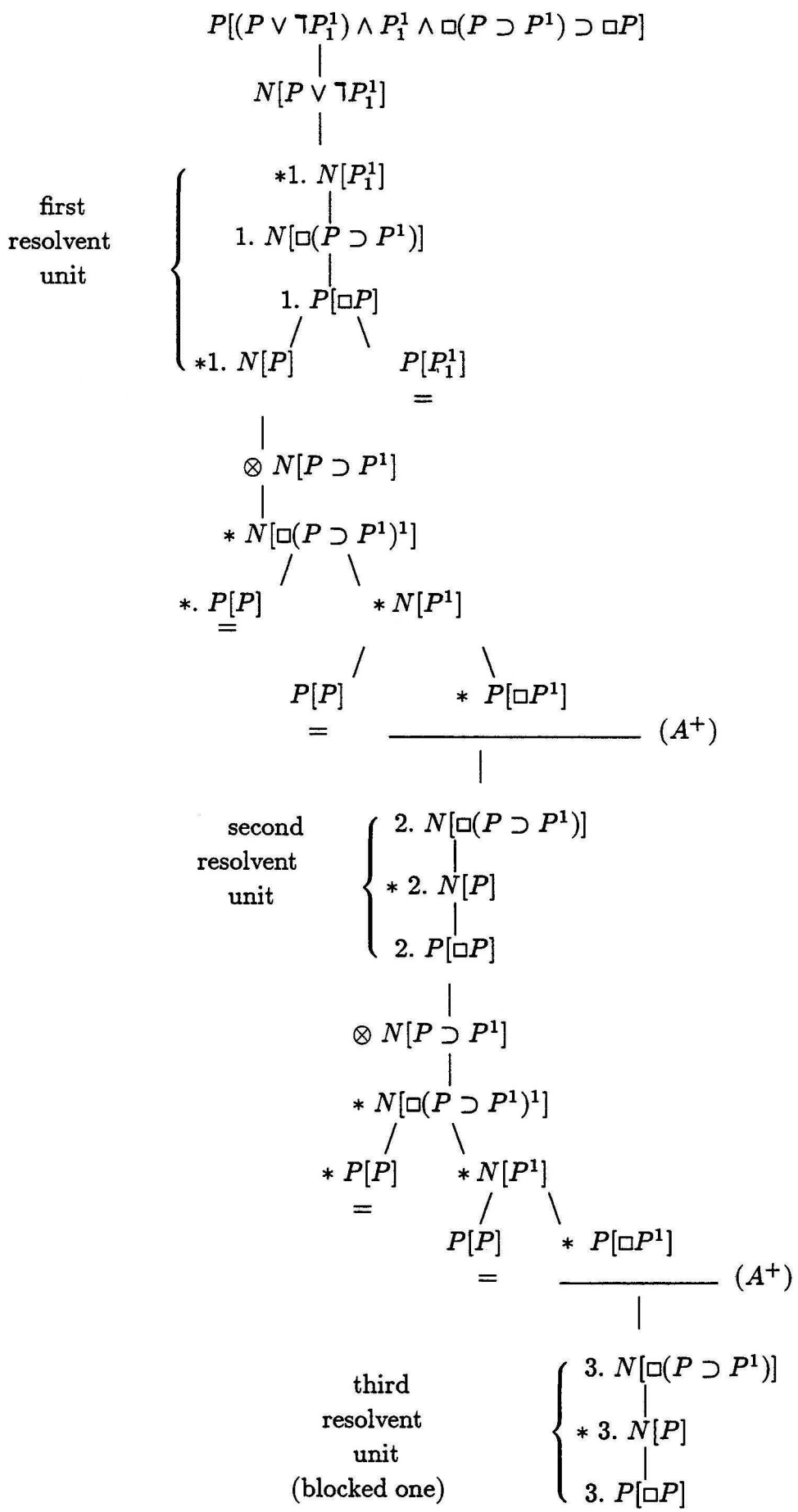
All the rules of inference, except for $(\Box)$ and (A), are "local" transformations with logical operators and the temporal symbol $\Box$. Each "local" rule of inference

may be applied to any vertex of a branch of the tree. The rules of inference ($\Box$) and (A) are "non-local". They are applied to some specified vertexes. To define the set of these vertexes and to define the notion of a saturated branch, let us introduce three sorts of marked vertexes. Let A be a member of a quasiprimary sequent (see Section 3) and let $P[A]$ not belong to the "logical" branch of $(\rightarrow \Box^1)$, then the vertex of the form $\delta[A]$ ($\delta \in \{N, P\}$) will be called a resolvent vertex and denoted by $k\delta[A]$, where $k \in \omega$. All resolvent vertexes between the applications of (A) will be denoted by the same natural number. Let $k\delta[A_1], \ldots, k\delta[A_n]$ be all the resolvent vertexes with the same k. Then the set of vertexes $k.\delta[A_1], \ldots, k.\delta[A_n]$ will be called the k-th resolvent unit. Let A be a member of a primary sequent (see Section 3), then the vertex of form $\delta[A]$ ($\delta \in \{N, P\}$) will be called a primary vertex and denoted by $*\delta[A]$. Let us note that any resolvent vertex $k\delta[A]$ (where $A \neq \Box B$ and $B \neq B_1^1$) will be a primary vertex. Let a vertex $\delta[A]$ be the son of the resolvent vertex and $\delta[A]$ be not a primary vertex, then $\delta[A]$ will be called a candidate to the primary vertex and will be denoted by $\circledast\delta[A]$. All sons of the vertex $\otimes\delta[A]$ will be either the candidate to the primary vertex or the primary vertex. Let us generate all possible primary vertexes and then let us apply (A) to them. All vertexes related to the temporal premise of (A) compose a $(k+1)$-st resolvent unit, if the previous resolvent unit was the k-th one in the branch. If in all the branches of the tree we get the resolvent unit of the shape $k.N[\Box A_1], \ldots, k.N[\Box A_n]$, $k.P[\Box A]$, then we do not generate primary vertexes but apply ($\Box$) to these resolvent units. Let $U_k = N[A_1], \ldots, N[A_n], P[B_1], \ldots, P[B_m]$ be a k-th resolvent unit, then the sequent $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ will be called a sequential image of U_k and denoted by $S[U^k]$. We say that the unit U_k absorbs the unit U_l (in symbols $U_k \succ U_l$) if $S[U_k] \succ S[U_l]$ (see Section 3). Let U_k and U_l be k-th and l-th ($k < l$) resolvent units and U_k, U_l belong to the same branch at the tree. We say that U_k and U_l are saturated and U_l is absorbed if $U_k \succcurlyeq U_l$. We say that the k-th resolvent unit U_k is blocked if (1) U_k is absorbed and (2) $\forall j$ $(f \leqslant j < k-1)$ U_j is saturated A branch of the tree will be called saturated if it contains a blocked resolvent unit. A branch of the tree D will be called logical closed if it contains two vertexes $P[E(t_1, \ldots, t_n)]$ and $N[E(p_1, \ldots, p_n)]$, where terms t_i and p_i $(i = 1, \ldots, n)$ are unifiable (see e.g. [2]) by the set of terms from D. (For simplicity of unification process all pure logical formulas (i.e. without $\Box$ and indices) we shall consider as skolemized ones). A branch in the tree D will be called clossed if it is logical closed or saturated. The tree D will be called clossed if all the branches of the tree D are closed. The notation $T\mathrm{Sat} \vdash A$ means that one can construct the closed tree in TSat of the formula A.

Theorem 4.1. $G_{L\omega} \vdash S \Leftrightarrow T\mathrm{Sat} \vdash P[S^F]$, where S is an M-sequent.

Proof: using Theorem 3.4 and [2].

Example 4.1 (a) Let $S = (P \vee \neg P_1^1), P_1^1, \Box(P \supset P^1) \rightarrow \Box P$, then $S^F = (P \vee \neg P_1^1) \wedge P_1^1 \wedge \Box(P \supset P^1) \supset \Box P$. Let us construct a closed tree of S^F in TSat^+, where TSat^+ is the calculus obtained from TSat replacing (A) by (A^+) (see Section 3):


$P[(P \vee \neg P_1^1) \wedge P_1^1 \wedge \Box(P \supset P^1) \supset \Box P]$
$N[P \vee \neg P_1^1]$
first resolvent unit
$*1.\ N[P_1^1]$
$1.\ N[\Box(P \supset P^1)]$
$1.\ P[\Box P]$
$*1.\ N[P]$
$P[P_1^1]$
$\otimes\ N[P \supset P^1]$
$*\ N[\Box(P \supset P^1)^1]$
$*.\ P[P]$
$*\ N[P^1]$
$P[P]$
$*\ P[\Box P^1]$
(A^+)
second resolvent unit
$2.\ N[\Box(P \supset P^1)]$
$*\ 2.\ N[P]$
$2.\ P[\Box P]$
$\otimes\ N[P \supset P^1]$
$*\ N[\Box(P \supset P^1)^1]$
$*\ P[P]$
$*\ N[P^1]$
$P[P]$
$*\ P[\Box P^1]$
(A^+)
third resolvent unit (blocked one)
$3.\ N[\Box(P \supset P^1)]$
$*\ 3.\ N[P]$
$3.\ P[\Box P]$

(b) Let us consider the Example from [7], i.e. $S = \Box P \wedge \neg\Box\neg\neg P \rightarrow$. In the closed tree, as indicated below, one can apply ($\Box$) instead of generating $Re^k(S^F)$ (in [7] for this example the rule of inference ($\rightarrow \Box$) implicitly is used):

$$
\begin{array}{c}
N[\Box P \wedge \neg\Box\neg\neg P] \\
| \\
N[\Box P] \\
| \\
P[\Box\neg\neg P] \\
| \\
\hline \\
P[P] \\
| \\
N[P] \\
| \\
N[\Box P^1] \\
=
\end{array}
\qquad (\Box)
$$

(c) Let S be the same as in Example 2.1(a), then $S^F = P(c) \wedge \Box\Omega_1 \wedge \Box\Omega_2 \wedge \Box\Omega_3 \supset \Box Q(c)$. Then applying the rules of inferences for logical operators and $(\Box^1), (A)$ to $P[S^F]$ we get a closed tree, containing a saturated branch with the following blocked resolvent unit: $N[\forall x P(f(x)),\ \Box\Omega_1,\ \Box\Omega_2,\ \Box\Omega_3],\ P[Q(c)]$.

Remark 4.1. For arbitrary miniscoped formulas (i.e in temporal parts of which quantifiers enter only the formulas of the form $Q\overline{x}E(Q \in \{\forall, \exists\})$, where E is an atomic formula) the rule of inference ($\Box$) must be replaced by the following one:

$$
\begin{array}{c}
N[\Box\Gamma],\ P[\Box A_1], \ldots, P[\Box A_n] \\
\diagup \qquad\qquad \diagdown \\
N[\Box\Gamma], P[A_1], P[\Box A_2], \ldots, P[\Box A_n] \qquad\qquad N[\Box\Gamma], P[\Box A_1], \ldots, P[\Box A_{n-1}], P[A_n]
\end{array}
$$

References

[1] M.Fisher, A resolution method for temporal logic, Proc. of IJCAI, Sydney 1991.

[2] M.Fitting, First-order Logic and Automated Theorem Proving. Springer-Verlag, 1990.

[3] J.H.Gallier, Logic for computer science: Foundations of automatic theorem prover, Harper and Row, New York, 1986.

[4] H.Kawai, Sequential calculus for a first order infinitary temporal logic, Zeitshr. für Math. Logic und Grundlagen der Math. **33**, (1987) 423–432.

[5] R.Pliuškevičius, Completeness criterion for Horn-like first order linear temporal logic. Proc. of conference on applied logic.Logic at Work, Amsterdam 1992.

[6] G.Sundholm, Systems of deduction. In D.Gabbay and F.Guenthner (eds.). Handbook of philosophical logic, **1**, D.Reidel, 1983, 133-188.

[7] P.Wolper, The tableaux method for temporal logic: an overview, Log. et anal. **28**, (1985), 119-136.

Deduction with First-order BDDs

Joachim Posegga **Klaus Schneider**

Universität Karlsruhe
Institut für Logik, Komplexität und Deduktionssysteme
Institut für Rechnerentwurf und Fehlertoleranz
Postfach 6980, 7500 Karlsruhe, FRG
Email: {posegga|schneide}@ira.uka.de

March 23, 1993

1 Introduction

Binary Decision Diagrams (BDDs) and variants of them are known in Computer Science since many decades and have been successfully applied in many fields. Especially experience in hardware verification (see e.g. (Brace *et al.*, 1990)) has shown that BDDs are very suited as an underlying datastructure for proving properties of propositional formulæ. Especially ordered BDDs[1] are well suited for deduction with propositional formulæ and can be implemented very efficiently (Bryant, 1986; Bryant, 1992). In the following, we will outline how these ordered BDDs can be extended to handle full first order logic.

As Automated Deduction is another research field interested in proving properties of logical formulæ, one would expect that BDDs have also been investigated there. Surprisingly, this it not at all the case, although there are some good arguments suggesting that it is worth doing this. BDDs have proven to be a good choice for reasoning about propositional logic, so carrying their working principle forward to the first-order case seems promising from the Automated Deduction's perspective.

Historically, there are only two papers in the mid-60s that treat non-ordered BDDs as a tool for theorem proving: Ehrenfeucht and Orłowska (1967) describe a propositional calculus and Orłowska (1969) extends it to a decidable subset of first-order logic. Recently, Posegga and Ludäscher (1992) proposed Shannon graphs, a variant of BDDs, for representing quantifier-free formulæ of first-order logic. The approach was carried forward from quantifier-free to general formulæ (Posegga, 1993) and an implementation of the method showed very good performance.

Shannon graphs bridge a gap between BDDs and semantic tableaux: on one hand, they can be understood as non-ordered BDDs, and on the other hand, they provide a linear representation of fully expanded tableaux with lemmata[2]. This linearity (w.r.t. the negation normal form of the formula) is due to the fact that Shannon graphs provide a joint representation for models and counter models of a formula, which cannot be achieved in tableaux.

Shannon graphs do not use orderings on atoms, since this would not be compatible with introducing free variables during the proof search, as free-variable tableaux do: such an ordering could be destroyed by the application of substitutions and an expensive re-ordering of the graphs would be necessary. This abstract outlines how the basic deduction mechanism of Shannon graphs can be carried forward to ordered BDDs — if the use of free variables is sacrificed. Alternate approaches to this have been described by Goubault (1993) and Billon (1991), but those require quantifier-free formulæ, which is not optimal for efficiently carrying out the proof search.

[1] Ordered BDDs provide an ordering on atoms that is used to eliminate redundancy during the construction of a BDD, and results in a unique normal form for propositional logic.

[2] See D'Agostino (1992) for tableau with lemmata.

2 The Calculus

For the sake of simplicity we will only consider formulæ in Skolemized negation normal form with properly renamed variables[3]. We assume the existence of a function *bdd* which maps a propositional formula into a BDD. We also assume the existence of standard logical operations on BDDs like conjunctively and disjunctively combining two BDDs and negating a BDD[4]. These operations will be denoted by the functions $bdd_\wedge$ and $bdd_\vee$, respectively.

The basic idea of our calculus is to embed "standard" BDDs in a first-order calculus. We use the function *f2bdd* on a first-order formula, by assuming that a formula is always treated as a propositional one; to achieve this, we simply regard first-order atoms and quantified formulæ as propositional atoms[5]. This can be formalized by defining an injective mapping from the set of all first–order atoms to a set of propositional ones. The mapping does not treat variables and function symbols appearing in an atom in a special way, but simply regards an atom as a string.

Furthermore, we use a special set of propositional atoms $\gamma_1, \gamma_2, \ldots$, that will be used to denote the scope of quantifiers in first-order formulæ, together with the variable that is universally quantified. Nodes in a BDD that are labeled with such atoms will be called γ-nodes; if γ_i is a γ-node, then $\mathcal{G}_i$ will denote the BDD representing the scope of the i^{th} quantifier in the input formula, and $\mathcal{V}_i$ denotes the quantified variable.

Definition 2.1 (First-order BDDs)
A BDD for a first-oder formula F is defined in the following way:

$$f2bdd(F) = \begin{cases} bdd(F) & \text{if } F \text{ is a literal} \\ bdd_\wedge(f2bdd(A), f2bdd(B)) & F = A \wedge B \\ bdd_\vee(f2bdd(A), f2bdd(B)) & F = A \vee B \\ bdd(\gamma_i) & \text{if } F = \forall x\, \Phi, \\ & \text{where } \mathcal{G}_i = f2bdd(\Phi) \text{ and } \mathcal{V}_i = x \end{cases}$$

The first graph ("Initial") of Figure 1 is an example first-order BDD for the formula

$$(\forall x\ P(x) \wedge Q(x)) \wedge \neg P(a) \wedge \neg Q(b)$$

and the ordering $\gamma_i < Pa < Pb < Qa < Qb$. The γ-node at the root of the BDD has been replaced by a node holding the BDD for the quantified subformula. All atoms appear without brackets in order to denote that they are actually treated as propositional atoms.

It should be easy to see that the function *f2bdd* preserves satisfiability of a formula F. Note, that an important property of propositional BDDs is necessarily missing in this framework: *f2bdd* does not produce a unique normal form for a first-order formula. This is a consequence of the fact that first-order logic is only semi-decidable.

In order to prove a propositional formula, the conversion into a reduced ordered BDD is sufficient as each unsatisfiable formula has the normal form **0**. In first order logic, however, this is not sufficient, as eventually copies of quantified formulæ are required. In order to obtain a proof procedure for first-order logic, we require an iteration step that alters $f2bdd(F)$ until unsatisfiability can eventually be detected by reducing the extension to **0**. The proposed procedure to achieve this is the following: a BDD containing a γ-node γ_i can be *extended* by replacing

[3]The calculus can be easily carried forward to general first-order formulæ.

[4]This corresponds to the standard functionality offered by existing implementations of BDD packages. The calculus will neither depend on a particular implementation, nor will it require special orderings, so we can ignore all these details.

[5]We will not use the notion "propositional variable" here to avoid confusion with variables in first-order formulæ.

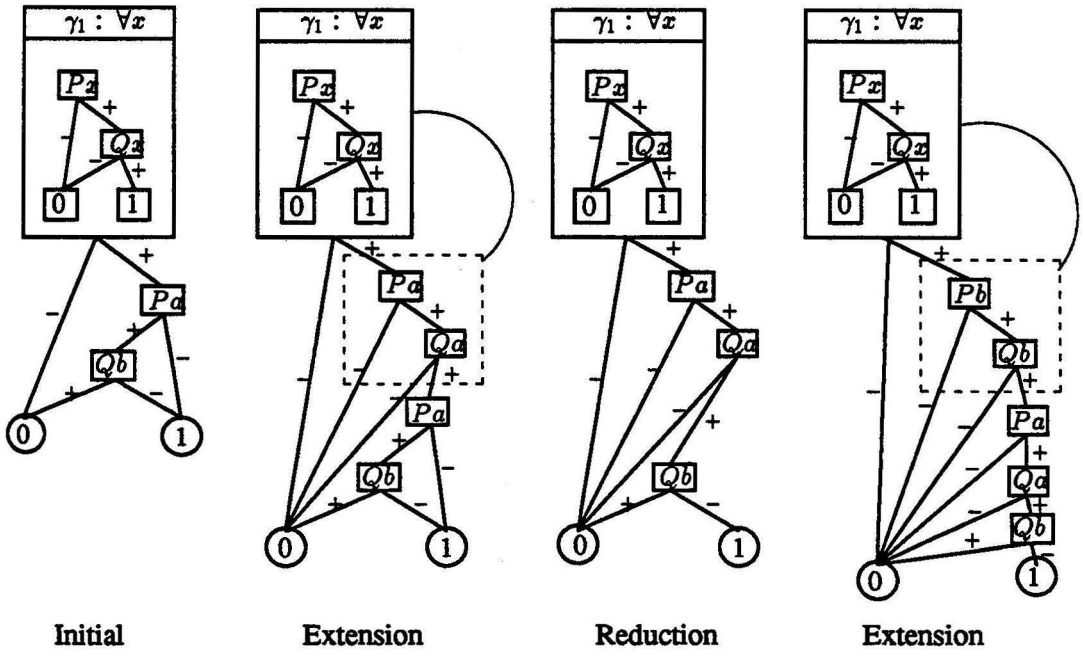


Figure 1: Examples of a First-order Proof with BDDs

γ_i with $\gamma_i \wedge (\mathcal{G}_i\sigma)$, where σ is a substitution mapping $\mathcal{V}_i$ in $\mathcal{G}_i$ to a ground term. Usually such an operation is provided by the composition operation of BDD-packages, so there is no need to actually modify a mechanism for handling standard BDDs.

Choosing the γ-node and selecting an appropriate substitution is of course a crucial point which is essential for the efficiency of the proof search. In general, it is desirable to choose σ such that the BDD becomes as small as possible after the extension, but also criteria of *fairness* have to be observed to guarantee completeness of an implementation.

The second graph in Figure 1 shows such an extension with the substitution $\sigma = [x/a]$. A subsequent reduction results in the third graph of the figure. After a second extension with the same γ-node and $[x/b]$, the resulting BDD (the forth graph) reduces to the single node **0**. This shows that the initial formula was unsatisfiable.

The correctness of extensions is again rather straightforward: an extension corresponds to replacing a formula $\forall x\ \Phi$ by $(\forall x\ \Phi) \wedge \Phi\sigma$, which is sound for any substitution σ. Completeness is less obvious, but given in the sense that there exists a finite sequence of extensions that result in an inconsistent BDD, if the initial formula was unsatisfiable (Herbrand's Theorem).

3 Conclusion

We have described a first-order proof procedure based on a standard, propositional BDD formalism. It works by keeping special nodes in BDDs that represent BDDs of quantified subformulæ. Such "quantified" BDDs are subsequently inserted into the superior BDD until a sufficient number of distinct ground instances of them have been produced in order to yield a propositionally inconsistent BDD.

A disadvantage of the approach is that free variables must not be used during the proof search.

Experience with semantic tableau has shown that this very advantageous, since the instantiation of quantified variables can then be delayed until contradictions arise. Deduction with non-ordered Shannon graphs (Posegga, 1993) therefore offers a more elegant solution for first-order deduction. The efficiency of propositional BDDs might, to some extend, weight this up, an experimental implementation for investigating this is under development.

Independently of the outcome of this, the approach presented here offers an elegant solution for representing quantifiers in BDDs, which is an important contribution: it shows how to lift ordered BDDs to(wards) first-order logic without loosing efficiency in the propositional case. This is advantageous for applications of BDDs, where propositional logic is mostly sufficient, but first-order constructs are useful in some cases. As an alternate approach, one could try to integrate free variables and unification into the algorithms underlying propositional BDDs. This, however, required a complete re-design, since currently known algorithms rely on the principle that local changes do not have global side-effects. Applying a substitution contradicts to this, and it is questionable whether this principle can be sacrificed without destroying the basis of efficiency of BDDs.

References

Jean-Paul Billon. A new approach of theorem proving for non clausal first order logic with equality based on generalized shannon's decomposition principle. Technical Report ORDA/DMA/91037, Bull Corporate Research Center, Paris, France, December 1991.

Karl S. Brace, Richard L. Rudell, & Randal E. Bryant. Efficient implementation of a BDD package. In *Proc. 27th ACM/IEEE Design Automation Conference*, pages 40 – 45. IEEE Press, 1990.

Randal Y. Bryant. Graph–based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C–35:677 – 691, 1986.

Randal Y. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. Technical report, Carnegie Mellon University. School of Computer Science, 1992.

Marcello d'Agostino. Are tableaux an improvement on truth–tables? Cut–free proofs and bivalence. *Journal of Logic, Language and Information*, 1(3), 1992.

A. Ehrenfeucht & E. Orłowska. Mechanical proof procedure for propositional calculus. *Bull. de L'Acad. Pol. des Sci., Série des sci. math., astr. et phys.*, XV(1):25–30, 1967.

Jean Goubault. Syntax independent connections. In *Proc. Second Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Marseilles, France, April 1993. published by Max-Planck-Institut für Informatik, Saarbrücken, Germany (technical report).

Ewa Orłowska. Automatic theorem proving in a certain class of formulae of predicate calculus. *Bull. de L'Acad. Pol. des Sci., Série des sci. math., astr. et phys.*, XVII(3):117 – 119, 1969.

Joachim Posegga & Bertram Ludäscher. Towards first-order deduction based on shannon graphs. In *Proc. German Workshop on Artificial Intelligence*, LNAI, Bonn, Germany, 1992. Springer.

Joachim Posegga. Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe. Dissertation, Universität Karlsruhe, FRG, 1993.

Experiments in Computing Prime Implicants / Implicates Using Techniques Not Requiring Clause Form

Extended Abstract

Anavai Ramesh *Neil V. Murray*

Inst. for Programming & Logics
Dept. of Computer Science
State Univ. of N.Y. at Albany
Albany, NY 12222

rameshag@cs.albany.edu nvm@cs.albany.edu

1. Introduction

Consequences expressed as minimal clauses that are implied by a formula are its *prime implicates*; minimal conjunctions that imply a formula are its *prime implicants*. Implicates are useful in certain approaches to non-monotonic reasoning [7,9], where all consequences of a formula set (e.g., the support set for a proposed commonsense conclusion) are required. The implicants are useful in situations where satisfying models are desired, as in error analysis during hardware verification.

In [8], we described several algorithms that employed dissolution for the purpose of computing the prime implicants and the prime implicates of a propositional formula. A key notion in that development is that of a linkless formula in *negation normal form* (NNF). Other algorithms for computing prime implicants and prime implicates have been developed by Slagle, Chang, and Lee [10], Jackson and Pais [1], Jackson [2], and by Kean and Tsiknis [3].

We describe experiments in which the non-clausal algorithm PI from [8] is employed to compute the prime implicates of an arbitrary propositional formula. (Computing prime implicants is the dual problem: If C is a prime implicate of G, then $\neg$C is a prime implicant of $\neg G$.) The use of dissolution in conjunction with PI is optional; we experimented on PI both with and without dissolution. Our methods do not require input to be either in conjunctive or in disjunctive normal form. We show empirically that this is often an advantage. We also describe a class of formulas for which avoiding DNF (CNF) in the intermediate stage of computation results in an exponential speedup in computing prime implicates (implicants).

Our algorithms combine the path-based techniques from [1] with techniques that are inference-based as in [3]. We assume the reader to be familiar with our path semantics viewpoint, our graphical representation of formulas in classical logic, and path dissolution [4,6]. Proofs are omitted for lack of space and can be found in [8].

If we dissolve in an NNF formula (semantic graph) G until it is linkless, we call the resulting graph the *full dissolvent of* G and denote it by FD(G). The c-paths in FD(G) are exactly the satisfiable c-paths in G, and the consequences of G are represented in the d-paths of FD(G) (see Theorem 4 of [8]).

2. Using PI to Compute Prime Implicates

Theorem. Given a formula FD(G) in NNF, PI computes all and only the prime implicates of FD(G) (see [8]).

Since $\neg$PI($\{\varnothing\}$, $\neg G$) = FD(G), the prime implicates of a formula G can be computed with either Option 1 or Option 2 using PI:

† This research was supported in part by National Science Foundation Grant CCR-9101208.

Option 1

1. Compute G', the full dissolvent of G (with respect to c-links)
2. If G' is empty
   then the only prime implicate is *false*
   else call PI($\{\emptyset\}$, G')

Option 2

1. Compute $G' = \neg$ PI($\{\emptyset\}$, $\neg G$)
2. If G' is empty
   then the only prime implicate is *false*
   else call PI($\{\emptyset\}$, G')

3. Experiments on NNF Formulas

The methods described above have been implemented and tested on randomly generated NNF formulas. A third option (containing two rounds of dissolution) performed poorly compared with Options 1 and 2 and is not discussed.

TABLE I Number of literal occurrences = 25

Alphabet size = 5		Alphabet size = 12		Alphabet size = 20	
Option 1 (with dissolution)	Option 2 (without dissolution)	Option 1 (with dissolution)	Option 2 (without dissolution)	Option 1 (with dissolution)	Option 2 (without dissolution)
1.1	0.6	1.9	5.2	2.3	5.2
0.9	0.6	0.8	1.7	3.0	16.4
0.9	0.5	1.5	0.0	1.5	16.5
0.9	0.6	2.7	1.0	6.5	54.2
0.8	0.6	2.1	0.7	1.9	7.1
0.7	0.5	1.0	8.0	2.0	8.2
0.8	0.5	0.9	1.9	0.8	0.9
0.9	0.6	1.3	.1	3.4	86.7
0.9	0.7	1.0	0.7	3.8	7.7
0.8	0.5	1.7	0.8	6.4	16.3

Number of literal occurrences = 50

Alphabet size = 5		Alphabet size = 25		Alphabet size = 45	
Option 1 (with dissolution)	Option 2 (without dissolution)	Option 1 (with dissolution)	Option 2 (without dissolution)	Option 1 (with dissolution)	Option 2 (without dissolution)
2.0	0.7	419.7	1613.2	918.1	4123.7
1.0	0.7	9.1	67.0	585.1	3964.3
1.0	1.1	318.9	4276.4	289.4	3133.4
0.9	0.5	62.6	19040.4	1469.8	5573.8
1.0	0.7	1098.4	2418.0	161.5	1081.3
0.8	0.9	724.4	10986.9	2111.5	1808.9
1.1	0.7	41.0	2214.5	1705.5	35185.9
0.9	0.9	1110.2	2140.5	842.6	> 10 hrs.
2.5	1.0	135.1	3293.9	1573.8	6627.4
0.9	0.5	2.5	20.4	897.8	38009.3

3.1. Implementation

The system is currently implemented in PASCAL, Common LISP (KCL) and C. The key parts of the system consists of DISSOLVER (in PASCAL) and PI (in LISP). The interface between the two is through a driver written in C. The complete system runs on a SUN 3/60.

Table I involves NNF formulas with 25 and 50 literal occurrences. The sub-cases are classified according to alphabet size; as the alphabet size decreases, the average number of links in a formula of fixed size increases.

Using dissolution (Option 1) is better on the average. The most improvement over Option 2 occurs in cases where the alphabet size is high. However Option 1 is not greatly inferior in cases involving smaller alphabets.

Note that the times are given in seconds; the runs were done on a SUN 3/60.

4. Formulas Intractable for CNF/DNF Based Approaches.

Consider the formula $\bigwedge_{i=1}^{N} (A_i = B_i)$. In NNF, each equivalence is of the form $((A_i \wedge B_i) \vee (\overline{A_i} \wedge \overline{B_i})$. The size of this formula is $\mathbf{O}(N)$. There are $2 \cdot N$ prime implicates of the formula; each is of the form $(A_i \vee B_i)$ or $(\overline{A_i} \vee \overline{B_i})$, $i \leq N$.

Using Option 1 we can get the set of prime implicates in $\mathbf{O}(N^2)$ time. The reason is that these formulas have no (conjunctive) links, and this can be verified in $\mathbf{O}(N^2)$ time. The unaltered non-DNF output of Step 1 is acceptable as input to PI in Step 2. As a result, the implicates are collected in $\mathbf{O}(N^2)$ time. Using Option 2 results in $\mathbf{O}(2^N)$ running time because in that case (and, e.g., in the algorithms of [1] and [10]) an intermediate DNF representation is created. Since the size of any DNF equivalent of the formula is $\mathbf{O}(2^N)$, the $\mathbf{O}(2^N)$ running time inevitably results.

Some algorithms do not create such an intermediate DNF representation, but instead generate all implicates directly through inference techniques such as resolution (e.g., [2]). The implicates of the formulas above can be computed efficiently using these techniques. However, in [5] we introduced a class of formulas $\Phi = \{\Phi_1, \Phi_2, \cdots\}$ for which conversion to either CNF or to DNF is expensive. Each Φ_i is unsatisfiable, contains $2 \cdot i^2$ literals, and has a minimal CNF equivalent consisting of $n^2 + n^{n+1}$ literals. The DNF equivalent is the empty disjunction, but this can only be verified by discovering that all $n + n^n$ disjuncts have a link and can be deleted.

Let M_i be a literal not occurring in Φ_i and consider the formula $\Phi_i' = \Phi_i \vee M_i$. Obviously, Φ_i' has $\{M_i\}$ as its only implicate, but any method based on either CNF or on DNF would find Φ_i' intractable. Nevertheless, dissolution can handle Φ_i and hence Φ_i' in polynomial time [5]. Thus Option 1 can compute implicates for these formulas efficiently.

5. Conclusions

These results suggest that the use of one round of dissolution in conjunction with PI often results in much better performance than when PI is used twice, in computing prime implicates. The main reason seems to be that PI can accept NNF input that is often more concise than its DNF equivalent, and that dissolution often produces just such concise NNF as output. In those cases where dissolution does not help, its use usually does not result in a significant penalty.

Our empirical results are augmented by the discovery of the class of formulas described in Section 5, for which intermediate normalization is shown to result in an exponential penalty.

References

1. Jackson, P., and Pais, J. Computing Prime Implicants. *Proceedings of the 10th International Conference on Automated Deduction*, Kaiserslautern, W. Germany, July, 1990. In *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 449, 543-557.

2. Jackson, P. Computing prime implicants incrementally. *Proceedings of the 11th International Conference on Automated Deduction*, Saratoga Springs, NY, June, 1992. In *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 607, 253-267.

3. Kean, A. and Tsiknis, G. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation* **9** (1990), 185-206.

4. Murray, N.V., and Rosenthal, E. Path dissolution: A strongly complete rule of inference. *Proceedings of the 6th National Conference on Artificial Intelligence*, Seattle, WA, July 12-17, 1987, 161-166.

5. Murray, N.V., and Rosenthal, E. Reexamining Intractability of Analytic Tableaux. *Proceedings* of the *1990 International Symposium on Symbolic and Algebraic Computation*, Tokyo, Japan, Aug. 20-24, 1990, 52-59.

6. Murray, N.V., and Rosenthal, E. Dissolution: Making paths vanish. To appear, *J.ACM*.

7. Przymusinski, T.C. An algorithm to compute circumscription. *Artificial Intelligence* **38** (1989), 49-73.

8. Ramesh, A., Becker, G., and Murray, N.V. On computing prime implicants and prime implicates. Presented at the Workshop on Theorem Proving with Analytic Tableaux and Related Methods, March 18-20, 1992, Lautenbach, Germany. Technical Report TR 92-3, Dept. of CSI, SUNY Albany, Albany, NY 12222.

9. Reiter, R. and de Kleer, J. Foundations of assumption-based truth maintenance systems: preliminary report. *Proceedings of the 6th National Conference on Artificial Intelligence*, Seattle, WA, July 12-17, 1987, 183-188.

10. Slagle, J.R., Chang, C.L., and Lee, R.C.T. A new algorithm for generating prime implicants. *IEEE Transactions on Computers,* **C-19(4)** (1970), 304-310.

Extended Tableaux for Specification Refinement

Pedro Savadovsky
Department of Computing
Imperial College
University of London
ps@doc.ic.ac.uk

13 March 1993

Abstract

We present an extension of Analytic Tableaux to solve a class of implementation problems in software engineering. This extension, *definition tableaux*, is proved correct and complete relative to existence of solutions for *implementations by definitions*. An implementation by definitions is an interpretation of a source theory in a conservative extension of a target theory. Given a pair of source and target theories in first-order languages, a relativization predicate and a translation of the source equality predicate, there exist an implementation by definitions iff there exist a closed definition tableau for this implementation problem instance. Building a closed definition tableau, one deduces the correct definitions in the target language for the other source predicate and function symbols.

1 Introduction

It has been proposed as a general model of software development, the view of specifications as theories in a linguistic system and implementation as linguistic transformations [TM87]. Previous work has shown that refinement by *canonic steps* gives a clear theoretical foundation for many software development paradigms [LST83]. While the theory behind these development models seems well established, little has been done in the way of foundations for development tools that take advantage of the canonic step paradigm. In this paper we present an extension of the tableaux method as a foundation for semi-automatic tools to help the designer in the refinement process.

Specification refinement in this paradigm is seen as a series of canonic steps. At each step one has a *source* specification that is to be implemented in some extension of a *target* specification. The source specification is the presentation of a theory T_s in a logic L_1, and the target specification, the presentation of a theory T_t in a logic L_2. In this paper we consider the simplified case where L_1 and L_2 are first-order logic with equality and function symbols.

A canonic step is defined as an interpretation of theory T_s in a theory T_t', where T_t' is a *conservative extension* of the target theory.

2 Implementation by Definitions

Implementation by definitions (IBD) is a class of specification refinements in first order logic in which the specification of the target (abstract) machine is extended by a set of definitions for the translated symbols of the source specification. Extensions by definitions of a theory are those where one adds to the theory presentation a set of defining axioms for new function and predicate symbols. A defining axiom for a function symbol f is a formula:

$$\forall x \forall y \; X(x,y) \equiv (f(x) = y)$$

where $X(x,y)$ is a formula in the target language that denotes the graph of a function[1]. A defining axiom for a predicate symbol P is a formula:

$$\forall x \; X(x) \equiv P(x)$$

It is well-known that this kind of extension is conservative [Sho67].

An instance of an IBD problem is a pair of source and target specifications, together with a non-empty relativization predicate ρ. A relativization predicate is defined as a formula of the target language, that holds true for those values of variables of the target language that are legal representations of variables of the source language. Intuitively, one is mapping the elements of the source domain into elements of the target domain [TM87].

A *solution* for an IBD problem is a set of definitions D such that there is an interpretation of the theory T_s in the extension $T_t \cup D$.

One way to find solutions for IBD problems is as follows:

1. Extend the target language with the function and predicate symbols of the source theory, using renaming if necessary.
2. This induces a translation τ of the formulae of the source theory in the target language. We assume that a (non-empty) relativization predicate ρ is given, as well as a translation of the source equality predicate.
3. Next find a set of definitions D for the new symbols such that the following formulae are theorems of the target theory:

    1. $A_t \rightarrow (D_\Lambda \rightarrow \tau(A_s))$
    2. $(A_t \wedge D_\Lambda) \rightarrow \forall x \; (\rho(x) \rightarrow \rho(f)(x))$ for each new function symbol f

    where A_t and A_s are the conjunctions of the target and source theories, and D_Λ is the conjunction of the definitions.

Using the interpretation theorem [Sho67] one can prove that a set of definitions D which satisfies these conditions is a solution for the given IBD problem [Sav92].

[1] Called the *definiens*. The formula $f(x) = y$ is the *definiendum*.

If we let the definiens X_i in definitions $d_i \in D$ be second-order formula *variables*, then to solve an IBD problem amounts to prove the validity of the formulae:

1. $A_t \rightarrow \exists X_1 \ldots \exists X_n \ (D_\Lambda \rightarrow \tau(A_s)$
2. $(A_t \wedge \exists X_1 \ldots \exists X_n \ D_\Lambda) \rightarrow \forall x \ (\rho(x) \rightarrow \rho(f)(x))$ for each new function symbol f

for a given IBD problem instance.

Fortunately one does not need to employ second-order proof procedures to solve IBD problems. IBD's have the property that if there is a solution then there is one in which the *definiens* X_i are related in a fixed way to subformulae of the target axioms. In fact, they are *interpolants* of the formula

$$A_t \rightarrow (D_\Lambda \rightarrow \tau(A_s))$$

that satisfy the condition (2) above [Sav92].

This suggests a theorem-proving framework for procedures to solve IBDs. Following Fitting's [Fit90] suggestion on finding interpolants constructively, we extend Smullyan's [Smu68] constructive proof of Craig's lemma to first-order logic with equality. We also add rules to unify the definiens X_i with subformulae of A_t, producing definitions for the new symbols in the target language. The resulting extended tableaux method is called *definition tableaux* [Sav92].

3 Symmetric Tableaux

Symmetric Tableaux are the tableaux counterparts of Smullyan's symmetric sequent calculus $\mathcal{S}$ [Smu68]. They are useful in our context mainly as an aid to prove correctness and completeness of definition tableaux. For the purpose of implementing actual refinement procedures, one could extend analytic tableaux directly.

The rules of the symmetric system $\mathcal{S}$ are:

$$\frac{K, Z \vdash L}{K, \neg\neg Z \vdash L}\ \neg\neg_l \qquad \frac{K \vdash L, Z}{K \vdash L, \neg\neg Z}\ \neg\neg_r$$

$$\frac{K, \alpha_1, \alpha_2 \vdash L}{K, \alpha \vdash L}\ (\alpha_l) \qquad \frac{K \vdash L, \alpha_1 \quad K \vdash L, \alpha_2}{K \vdash L, \alpha}\ (\alpha_r)$$

$$\frac{K, \beta_1 \vdash L \quad K, \beta_2 \vdash L}{K, \beta \vdash L}\ (\beta_l) \qquad \frac{K \vdash L, \beta_1, \beta_2}{K \vdash L, \beta}\ (\beta_r)$$

$$\frac{K, \gamma(a) \vdash L}{K, \gamma \vdash L}\ (\gamma_l) \qquad \frac{K \vdash L, \delta(a)}{K \vdash L, \delta}\ (\delta_r)$$

$$\frac{K, \delta(a) \vdash L}{K, \delta \vdash L}\ (\delta_l) \qquad \frac{K \vdash L, \gamma(a)}{K \vdash L, \gamma}\ (\gamma_r)$$

with the proviso for the last two rules that a do not occur in

$$K \vdash L$$

The *axiom* schemas of the system $\mathcal{S}$ are:

$$K, X \vdash X, L$$
$$K, \neg X \vdash \neg X, L$$
$$K, \neg X, X \vdash L$$
$$K \vdash X, \neg X, L$$

where X is an *atomic* formula.

This system is invertible, sound and complete[2]. Clearly it obeys the subformula principle.

In a tableaux version of Smullyan's symmetric sequent calculus one needs to distinguish between formulae of the left side and of the right side of a sequent. What we do is to label formulae with **L** or **R**, for *left* and *right* respectively.

Here then are the symmetric tableaux expansion rules, where the indexes r and l indicate a uniform notation formula labelled with an **R** or an **L** respectively:

$$\frac{\mathbf{L}\neg\neg Z}{\mathbf{L}Z}\ \neg\neg_l \qquad \frac{\alpha_l}{\begin{array}{l}\alpha_{1l}\\ \alpha_{2l}\end{array}}\ \alpha_l \qquad \frac{\beta_l}{\beta_{1l} \mid \beta_{2l}}\ \beta_l \qquad \frac{\gamma_l}{\gamma_l(t)}\ \gamma_l \qquad \frac{\delta_l}{\delta_l(p)}\ \delta_l$$

$$\frac{\mathbf{R}\neg\neg Z}{\mathbf{R}Z}\ \neg\neg_r \qquad \frac{\beta_r}{\begin{array}{l}\beta_{1r}\\ \beta_{2r}\end{array}}\ \beta_r \qquad \frac{\alpha_r}{\alpha_{1r} \mid \alpha_{2r}}\ \alpha_r \qquad \frac{\gamma_r}{\gamma_r(t)}\ \gamma_r \qquad \frac{\delta_r}{\delta_r(p)}\ \delta_r$$

where Z is an unsigned formula, t is a closed term and p is a new parameter.

Having defined the symmetric tableaux expansion rules we can define symmetric tableaux, much as standard analytic tableaux in Fitting [Fit90]:

Definition 3.1 Let $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$ be a finite set of labelled formulae. Then

1. The one branch tree

$$\begin{array}{c}\mathbf{L}A_1\\ \mathbf{L}A_2\\ \vdots\\ \mathbf{L}A_n\\ \neg\mathbf{R}B_1\\ \vdots\\ \neg\mathbf{R}B_m\end{array}$$

[2]Soundness and completeness proofs are presented in [Smu68]. Inversion is a simple checking of the rules and is omitted here.

1. **L** $\forall x\ P(x) \supset Q(x)$
2. **R** $\forall x\ \neg Q(x) \supset \neg P(x)$
3. **R** $\neg Q(a) \supset \neg P(a)\ (2, \gamma_r)$
4. **L** $P(a) \supset Q(a)\ (1, \gamma_l)$
5. **R** $\neg\neg Q(a)\ (3, \beta_r)$
6. **R** $\neg P(a)\ (3, \beta_r)$
7. **R** $Q(a)\ (5, \neg\neg_r)$
8. **L** $\neg P(a)\ (4, \beta_l)$
9. **L** $Q(a)\ (4, \beta_l)$

Figure 1: Closed Symmetric Tableau

is a symmetric tableau for $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$.

2. If $\mathcal{T}$ is a symmetric tableau for $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$ and $\mathcal{T}^+$ is obtained from $\mathcal{T}$ by the application of a symmetric tableau expansion rule, then $\mathcal{T}^+$ is a tableau for $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$.

□

A branch of a symmetric tableau is called *closed* if it contains any of the following pairs of literals:

1. a literal $\mathbf{L}F$ and a literal $\mathbf{R}F$.
2. a literal $\mathbf{L}F$ and a literal $\mathbf{L}\overline{F}$.
3. a literal $\mathbf{R}F$ and a literal $\mathbf{R}\overline{F}$.

A symmetric tableau is closed if all its branches are closed.

Example 3.2 As an example of a closed symmetric tableau, we prove the sequent:

$$\forall x\ (P(x) \supset Q(x)) \rightarrow (\forall x\ \neg Q(x) \supset \neg P(x))$$

A closed symmetric tableau for the corresponding set of labelled formulae

$$\{\mathbf{L}\forall x\ (P(x) \supset Q(x)), \mathbf{R}\forall x\ \neg Q(x) \supset \neg P(x)\}$$

is shown in figure 1. The leftmost branch contains **L** $\neg P(a)$ in line 8, and **R** $\neg P(a)$ in line 6, thus it is closed. Similarly the rightmost branch contains **L** $Q(a)$ and **R** $Q(a)$, and so it is closed. □

3.1 Extension with Equality

Definition 3.3 A *variant* of a signed sequent $K \vdash L$ is any sequent obtained from $K \vdash L$ by transfering any number of formulas from one side to the other and at the same time changing their sign. □

A variant of an unsigned sequent is obtained *negating* a formula and then transfering it to the other side. An α formula becomes a β one, and vice versa.

Now we may extend the system $\mathcal{S}$ to deal with equality. This is interesting, because we have a choice between different possibilities.

A first sound and complete system is obtained adding equality *rules* to $\mathcal{S}$. However, to have a complete system, on the light of Smullyan's completeness proof, we must add *all variants* of the basic three kinds of equality rules.

Thus we have a first version of a symmetric system for languages with equality (and functions). Let us call it $\mathcal{S}_{=}$.

This system is both sound and complete. It is invertible also, but does not obey the subformula principle.

Another option to extend $\mathcal{S}$, is to add to the *axioms* all the variants of equality axioms as done by Gallier [Gal86]:

$$
\begin{array}{l}
K \vdash t \approx t, L \\
K, (s_1 \approx t_1) \wedge \ldots \wedge (s_n \approx t_n) \vdash f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n), L \\
K, (s_1 \approx t_1) \wedge \ldots \wedge (s_n \approx t_n) \wedge P(s_1, \ldots, s_n) \vdash P(t_1, \ldots, t_n), L
\end{array}
$$

and all their variants, e.g.

$$
\begin{array}{l}
K, \neg(t \approx t) \vdash L \\
K, (s_2 \approx t_2) \wedge \ldots \wedge (s_n \approx t_n) \vdash \neg(s_1 \approx t_1), f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n), L \\
\vdots
\end{array}
$$

It turns out that to have completeness for this system we also have to add a *cut rule* to the rules of $\mathcal{S}$:

$$
\frac{K \vdash L, A \qquad M \vdash N, \neg A}{K, M \vdash L, N}\ Rcut
$$

The system with the axioms and rules of $\mathcal{S}$, extended with the equality axioms and their variants above, plus the cut rule, is both sound and complete.

We call this system $\mathcal{SK}_{=}$ because it is an extension of $\mathcal{S}$, but the almost the same system is studied by Gallier [Gal86] where it is called $G1^{nnf}_{=}$. The system $G1^{nnf}_{=}$ is defined for sequents with sets of formulae in negation normal form (nnf).

Now we might define symmetric tableaux with equality using the calculus $\mathcal{S}_{=}$. However we will instead define symmetric tableaux with equality based on the system $\mathcal{SK}_{=}$. This is because we will need a tableaux cut rule to introduce definiens for function symbols.

Unlike a tableaux system based on $\mathcal{S}_{=}$, our symmetric tableaux based on $\mathcal{SK}_{=}$ do not have equality rules. They have instead a corresponding number of cases in the closing rule.

Definition 3.4 The expansion rules for symmetric tableaux with equality are those for symmetric tableaux, plus the cut rule:

$$
\frac{}{Z \vee \neg Z}
$$

where Z is an atomic unsigned formula. □

The definition of symmetric tableaux with equality is the same as that of symmetric tableaux. The definitions of closed branch and tableau are:

Definition 3.5 (Closing Rule) A branch of a symmetric tableau with equality is called *closed* iff

- It contains any of the following pairs of literals:
  1. a labelled literal $\mathbf{L}F$ and a labelled literal $\mathbf{R}F$.
  2. a labelled literal $\mathbf{L}F$ and a labelled literal $\mathbf{L}\overline{F}$.
  3. a labelled literal $\mathbf{R}F$ and a labelled literal $\mathbf{R}\overline{F}$.
- All of the formulae in one of the sets below are in the branch:
  1. $\{\mathbf{R}(t \approx t)\}$ where t is any term,
  2. $\{\mathbf{L}(s_1 \approx t_1), \mathbf{L}(s_2 \approx t_2), \ldots, \mathbf{L}(s_n \approx t_n), \mathbf{R}(f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n))\}$ for any n-ary function symbol f and terms s_i, t_i,
  3. $\{\mathbf{L}(s_1 \approx t_1), \mathbf{L}(s_2 \approx t_2), \ldots, \mathbf{L}(s_n \approx t_n), \mathbf{L}P(s_1, \ldots, s_n), \mathbf{R}P(t_1, \ldots, t_n)\}$ for any n-ary predicate symbol P and terms s_i, t_i,
  4. or any *variant* of these sets obtained substituting a left by a right formula label, and vice versa.

A symmetric tableau with equality is closed if all its branches are closed. □

From the correspondence between symmetric tableaux with equality and the system $\mathcal{SK}_{=}$, it follows that it is a sound and complete tableaux system.

4 Definition Tableaux

Now we introduce a calculus for solving IBD's which is an extension of symmetric tableaux by special rules to make definitions. This calculus, *definition tableaux*, is proved correct and complete relative to solutions for IBD's.

How does one extend symmetric tableaux to solve our problem? The basic intuition is simple: one adds additional closing rules to symmetric tableaux, in which the formula variables are unified with formulae of the target language. In this way one simultaneously proves the condition stated above, and defines the new symbols.

We illustrate this idea with a propositional example, where there are no function symbols and the conditions for implementation of T_s in T_t reduce to

$$\vdash A_t \rightarrow (D_\Lambda \rightarrow \tau(A_s))$$

Example 4.1 Suppose we have a target theory with one extra-logical axiom:

$$(A \wedge B) \vee C$$

Let the new (propositional) symbols be P, Q, R and the translated source specification be the formula:

$$(P \vee Q) \supset R$$

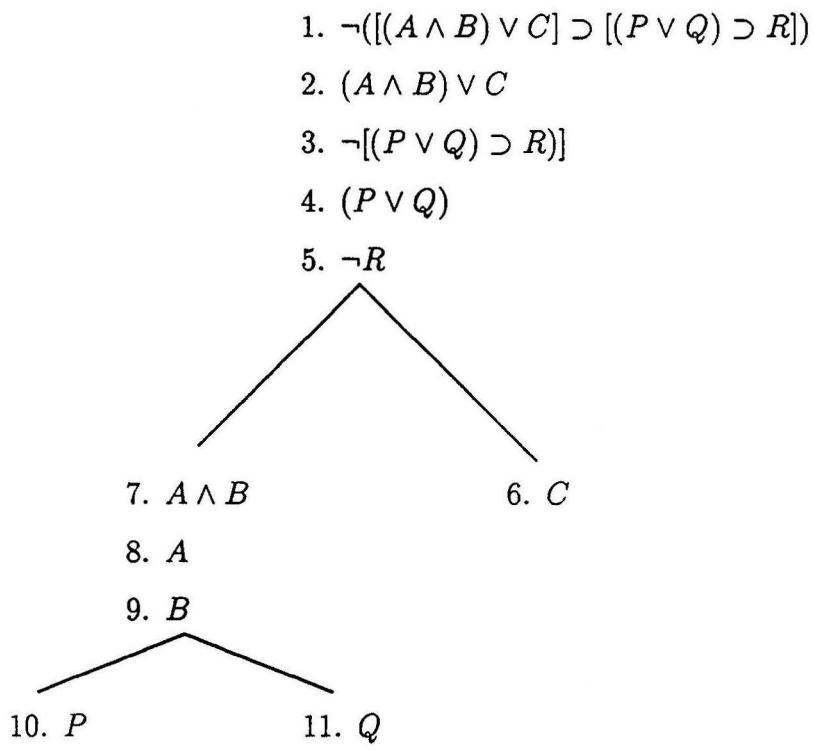


Figure 2: Tableaux expansion for the example

We want to find definitions

$$X_1 \equiv P$$
$$X_2 \equiv Q$$
$$X_3 \equiv R$$

for the new symbols, such that the formula:

$$(A \wedge B) \vee C \supset ([(X_1 \equiv P) \wedge (X_2 \equiv Q) \wedge (X_3 \equiv R)] \supset [(P \vee Q) \supset R])$$

is true.

Instead of proving this formula, we expand a tableau for the formula:

$$\neg(((A \wedge B) \vee C) \supset ((P \vee Q) \supset R))$$

obtained from the previous one deleting the conjunction of the definitions. A tableau expansion until there are only atomic formulae, is shown in figure 2. At the point shown in the figure we cannot continue the expansion, and the tableau is not closed. Now, the new symbols P, Q, R are undefined. Hence we choose meanings for them in such a way that all branches of the tableau become closed.

For example, the rightmost branch has the leaf labelled C in line 6. In line 5 we have $\neg R$. Now, if R was defined as C,

$$R \equiv C$$

then this branch would contain both $\neg C$ and C, hence would be closed.

In the same way we could close the leftmost branch if we defined:

$$P \equiv \neg A$$

because of lines 10 and 8, and the other branch would close with the definition:

$$Q \equiv \neg B$$

One can easily check that the interpretation condition is satisfied with these definitions.

This definition making rules follow the same basic pattern: if there is a formula $F(f)$ in the branch, containing some undefined symbol f and/or F, we look for a formula $P(d)$ of the target language and a substitution

$$\sigma = \{F/P, f/d\}$$

such that $\neg F(f)\sigma = \neg P(d)$. Then if we apply σ to the branch it will contain the pair $\neg P(d), P(d)$, thus being closed, and the substitution provides the required definitions for F and f.

To solve an IBD problem instance one tries to find a closed definition tableau for the formula

$$A_t \rightarrow \tau(A_s)$$

As we start a tableau expansion like that we have no definitions for the new symbols. Every time a definition is made in the above fashion, one adds this definition to a current *set of definitions*. When appliable, definitions in this set may be used to substitute function terms or predicates by their target language definiens. A final set of definitions is obtained if and when one manages to close the tableau. Thus the set of definitions is a kind of state in a definition tableau expansion. Also, all expansion and definition making rules must be labelled with appropriate sets of definitions in the premises and conclusions. Expansion rules will not introduce definitions, so the sets in the premises and conclusion are equal.

4.1 Definition Rules

Definition rules to assign new symbols to symbols of the target theory are obtained covering all possible cases. Basically one may have an atomic formula in the tableau, in which new symbols occur as:

- A predicate symbol without terms with undefined functions.
- A function symbol in an equation.
- A predicate symbol with terms with undefined functions.

These may be further classified in negated or un-negated formulae. We employ some notation to write these rules:

Notation 4.2 The *prefix* of a literal L is the empty string if L is atomic, and is the string '$\neg$' otherwise. We represent the prefix with the function $\bullet$:

$$\bullet(L) = \begin{cases} \textit{empty string} \text{ if } L \text{ is atomic,} \\ \text{'}\neg\text{' otherwise.} \end{cases}$$

We also denote the atomic part of a literal L by $|L|$: if $L = A$ or $L = \neg A$, with A atomic, then

$$|L| = A$$

□

Using this notation we have the equations $A = \bullet(A)|A|$ for any literal A.

Notation 4.3 Let $A = \bullet(A)|A|$ and $B = \bullet(B)|B|$ be a definiens and a definiendum. Then the function $\diamond$ is defined as:

$$\diamond A = \begin{cases} A \text{ if } \bullet(A) = \bullet(B), \\ \neg A \text{ if } \bullet(A) \neq \bullet(B) \end{cases}$$

□

Notation 4.4 Let X_1 be the definiens in the new definition $d_1 = (X_1 \equiv Y)$, introduced by a definition rule. Then

$$D_{i+1} = D_i \oplus \{X_1 \equiv Y\}$$

where $\oplus$ is defined as:

1. If there is no definition for Y in D_i then

$$D_i \oplus \{X_1 \equiv Y\} = D_i \cup \{X_1 \equiv Y\}$$

2. If $d_2 \in D_i$ is a definition $d_2 = (X_2 \equiv Y)$, then

   (a) If $\bullet(B) = \neg$ then

$$D_i \oplus \{d_1\} = D_i - \{d_2\} \cup \{(X_1 \vee X_2) \equiv Y\}$$

   (b) If $\bullet(B)$ is the empty string then

$$D_i \oplus \{d_1\} = D_i - \{d_2\} \cup \{(\neg X_1 \wedge \neg X_2) \equiv Y\}$$

□

4.2 Definition Tableaux

We are ready to define the expansion and the closing rules for definition tableaux. The expansion rules are simply the symmetric tableaux expansion rules as presented above, labelled with the corresponding sets of definitions.

$\dfrac{\mathbf{L}\neg\neg Z\ ;\ D_i}{\mathbf{L}Z\ ;\ D_i}$	$\dfrac{\alpha_l\ ;\ D_i}{\begin{array}{c}\alpha_{1l}\\ \alpha_{2l}\ ;\ D_i\end{array}}$	$\dfrac{\beta_l\ ;\ D_i}{\beta_{1l}\ ;\ D_i \mid \beta_{2l}\ ;\ D_i}$	$\dfrac{\gamma_l\ ;\ D_i}{\gamma_l(t)\ ;\ D_i}$	$\dfrac{\delta_l\ ;\ D_i}{\delta_l(p)\ ;\ D_i}$
$\dfrac{\mathbf{R}\neg\neg Z\ ;\ D_i}{\mathbf{R}Z\ ;\ D_i}$	$\dfrac{\beta_r\ ;\ D_i}{\begin{array}{c}\beta_{1r}\\ \beta_{2r}\ ;\ D_i\end{array}}$	$\dfrac{\alpha_r\ ;\ D_i}{\alpha_{1r}\ ;\ D_i \mid \alpha_{2r}\ ;\ D_i}$	$\dfrac{\gamma_r\ ;\ D_i}{\gamma_r(t)\ ;\ D_i}$	$\dfrac{\delta_r\ ;\ D_i}{\delta_r(p)\ ;\ D_i}$

Figure 3: Definition Tableaux Expansion Rules

Definition 4.5 (Definition Tableaux Expansion Rules) The expansion rules for definition tableaux are composed of:

- *tree building* rules shown in figure 3, plus the cut rule:

$$\frac{;D_i}{Z \vee \neg Z\ ;D_i}$$

  where Z is an atomic unsigned formula.

- *definition* rules, shown figures 4 and 5. □

Now we define the central concept of definition tableaux, just as was done for symmetric tableaux:

Definition 4.6 (Definition Tableaux) Let $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$ be a finite set of labelled formulae, with $A_i \in L_t$ and $B_i \in L_\Delta$. Assume that the predicate symbols that occur in B_i are new to L_t and undefined. Then

1. The one branch tree

$$\begin{array}{c}\mathbf{L}A_1\\ \mathbf{L}A_2\\ \vdots\\ \mathbf{L}A_n\\ \neg\mathbf{R}B_1\\ \vdots\\ \neg\mathbf{R}B_m\end{array}$$

   is a *definition tableau* for $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$ with set of definitions $D = \phi$.

2. If $\mathcal{T}$ is a definition tableau for $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$ and $\mathcal{T}^+$ is obtained from $\mathcal{T}$ by the application of a definition tableau expansion rule with a resulting set of definitions D^+, then $\mathcal{T}^+$ is a definition tableau for $\{\mathbf{L}A_1, \mathbf{L}A_2, \ldots, \mathbf{L}A_n, \mathbf{R}B_1, \ldots, \mathbf{R}B_m\}$ with set of definitions D^+.

□

Rule 1

Let $A \in L_t, Y \in L_\Delta$ and $P \in L_\Delta$ be literals, and $f \in L_\Delta$ a function symbol.

$$\frac{\mathbf{L}A(t), \mathbf{R}\neg Y(x)\,;\, D_i}{\mathbf{R}(\forall x\ \diamond A(x) \equiv Y(x))\,;\, D_{i+1}}\ r_a$$

where

$$D_{i+1} = D_i \oplus \{\forall x\ \diamond A(x) \equiv Y(x)\}$$

is a consistent set.

$$\frac{\mathbf{L}A(\underline{v}, w), \mathbf{R}\neg Y(\underline{x}, y)\,;\, D_i}{\mathbf{R}(\forall x \forall y\ \diamond A(\underline{x}, y) \supset Y(\underline{x}, w))\,;\, D_{i+1}}\ r_b$$

where

$$D_{i+1} = D_i \oplus \{\forall x \forall y\ \diamond A(x, y) \supset Y(x, y)\}$$

is a consistent set, $|Y(x, y)| = (f(x) = y)$, and
$A_t \vdash \forall x \forall y_1 y_2\ [A(x, y_1) \wedge A(x, y_2)] \supset (y_1 = y_2)$
$A_t \vdash \rho(w)$

$$\frac{\mathbf{L}A(t), \mathbf{R}\neg A(f(x))\,;\, D_i}{\mathbf{R}(\forall x(W(x,t) \supset (f(x) = t)))\,;\, D_{i+1}}\ r_d$$

where

$$D_{i+1} = D_i \oplus \{\forall x\ W(x,t) \supset (f(x) = t)\}$$

is a consistent set, $W(x,t)$ a conjunction of all formulae in the branch in which x or t occur (except $A(x)$), and
$A_t \vdash \forall x \forall y_1 y_2\ [(W(x,t) \supset f(x) = y_1) \wedge (W(x,t) \supset f(x) = y_2)] \supset (y_1 = y_2)$
$A_t \vdash \rho(t)$

$$\frac{\mathbf{L}A(t), \mathbf{R}\neg P(f(x))\,;\, D_i}{\begin{array}{l}\mathbf{R}(\forall x W(x,t) \supset f(x) = t) \\ \mathbf{R}(\forall x\ \diamond A(x) \equiv P(x))\,;\, D_{i+1}\end{array}}\ r_e$$

where

$$D_{i+1} = (D_i \oplus \{\forall x\ W(x,t) \supset f(x) = y\}) \oplus \{\forall x\ \diamond A(x) \equiv P(x)\}$$

is a consistent set, $W(x,t)$ is the conjunction of all formulae in the branch in which x or t occur (except $A(x)$), and
$A_t \vdash \forall x \forall y_1 y_2\ [(W(x,t) \supset f(x) = y_1) \wedge (W(x,t) \supset f(x) = y_2)] \supset (y_1 = y_2)$
$A_t \vdash \rho(t)$

Figure 4: Extended Definition Tableaux Rules

Rule 2

$$\frac{;D_i}{;D_i \oplus \{W(x,t_2) \supset Y(x,t_2)\}}\ r_c$$

where:

1. $D_i \cup \{W(x,t_2) \supset Y(x,t_2)\}$ is a consistent set,
2. there is a mating set $E = L \cup R \cup \{Y(x,y)\}$ in the branch, where
   (a)
   $$L = \{(s_1 = t_1), \ldots, (s_m = t_m)\} \neq \phi$$
   with $(s_i = t_i)$ being labelled with **L**,
   (b)
   $$R = \{(u_1 = v_1), \ldots, (u_n = v_n)\}$$
   with $(u_j = v_j)$ being labelled **R**,
   (c) $|Y(x,y)| = (f(x) = y)$,

   and there is a substitution σ that mates the set E, such that
   $$\sigma = \{f(x)/t_1, y/t_2, \ldots\}$$
   $$A_t \vdash \rho(t_2)$$
3. $W(x,t_2)$ is the conjunction of all formulae in the branch in which x or t_2 occur, with the exception of the formulae in E.

Figure 5: Extended Definition Tableaux Rules

Unlike analytic tableaux, definition tableaux are *syntetic*, in the sense of introducing arbitrary formulae with the cut rule. This is undesirable in a system for theorem proving, but it is necessary for introducion of function definiens.

The definitions of closed branch and tableau are:

Definition 4.7 (Closing Rule) A branch of a definition tableau with equality is called *closed* iff

- It contains any of the following pairs of literals:
  1. a labelled literal $\mathbf{L}F$ and a labelled literal $\mathbf{R}F$.
  2. a labelled literal $\mathbf{L}F$ and a labelled literal $\mathbf{L}\overline{F}$.
  3. a labelled literal $\mathbf{R}F$ and a labelled literal $\mathbf{R}\overline{F}$.
- All of the formulae in one of the sets below are in the branch:
  1. $\{\mathbf{R}(t \approx t)\}$ where t is any term,
  2. $\{\mathbf{L}(s_1 \approx t_1), \mathbf{L}(s_2 \approx t_2), \ldots, \mathbf{L}(s_n \approx t_n), \mathbf{R}(f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n))\}$ for any n-ary function symbol f and terms s_i, t_i,
  3. $\{\mathbf{L}(s_1 \approx t_1), \mathbf{L}(s_2 \approx t_2), \ldots, \mathbf{L}(s_n \approx t_n), \mathbf{L}P(s_1, \ldots, s_n), \mathbf{R}P(t_1, \ldots, t_n)\}$ for any n-ary predicate symbol P and terms s_i, t_i,
  4. or any *variant* of these sets obtained substituting a left by a right formula label, and vice versa.

A definition tableau with equality is *closed* if all its branches are closed. □

4.3 Correctness and Completeness

We use the terms *correctness* and *completeness* relative existence of a solution for an IBD problem. The method of definition tableaux is said to be correct if a closed definition tableau with a final set of definitions D implies that there is a solution D for a given IBD problem instance. It is complete if the existence of a solution D for the instance, implies the existence of a closed definition tableau with set of definitions D.

In this sense correctness and completeness of definition tableaux are proved by the theorems below. Complete proofs are presented in [Sav92].

To prove correctness we start mapping the rules of definition tableaux into rules of symmetric tableaux. Expansion rules correspond one to one. As for definition making rules, one maps them into sequences of symmetric tableaux expansion rules. Finally one maps a closed definition tableau for the *implementation formula*

$$A_t \rightarrow \tau(A_s)$$

with set of definitions D, into a symmetric tableaux proof for the formula

$$A_t \rightarrow (D_\Lambda \rightarrow \tau(A_s))$$

Function closure conditions are also verified, to obtain:

Theorem 4.8 (Correctness) Let Ψ be an IBD instance and $\mathcal{T}_D$ a closed definition tableau with a set of definitions D for the corresponding implementation formula

$$A_t \rightarrow \tau(A_s)$$

If every function $f(x)$ defined in D is defined for all values x in the domain, then there is an implementation by definitions of the source in the target theory. □

Completeness is proved in a similar fashion. One maps a symmetric tableaux proof of

$$A_t \rightarrow (D_\Lambda \rightarrow \tau(A_s))$$

into a definition tableaux proof of

$$A_t \rightarrow \tau(A_s)$$

with set of definitions D. To map definition applications in the symmetric proof one has to introduce the definiens in the definition tableau. This is done using the cut rule of definition tableaux. As before one verifies that function closure holds, to get:

Theorem 4.9 (Completeness) Let Ψ be an IBD instance D a set of definitions such that

$$\vdash A_t \rightarrow (D_\Lambda \rightarrow \tau(A_s))$$

and $I = \langle \Theta, \pi \rangle$. If the pair $\langle I, D \rangle$ is an implementation by definitions of T_s in T_t then there exists a closed definition tableau for the implementation formula

$$A_t \rightarrow \tau(A_s)$$

having a set of definitions $D' \subseteq D$. □

5 Conclusions

Following Smullyan's approach to prove Craig's lemma, we extended a symmetric variant of analytic tableaux to obtain a tableaux method for specification refinement. Definition tableaux suggest procedures to solve IBD problems, much like standard tableaux procedures as presented by Fitting [Fit90]. Such procedures augmented by efficient strategies are useful components in CASE environments based on the canonic step paradigm.

The input to such a procedure is a pair of theory presentations, a relativization predicate and a translation of the equality predicate. The procedure finds the definitions for the other translated predicate and function symbols.

References

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, 1990.

[Gal86] J.H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Harper & Row, 1986.

[LST83] M.M. Lehman, V. Stenning, and W.M. Turski. Another look at software design methodology. Technical Report 83/13, Department of Computing, Imperial College, 1983.

[Sav92] P. Savadovsky. *A Tableaux Based Procedure for Specification Refinement.* PhD thesis, Imperial College, University of London, 1992.

[Sho67] Joseph R. Shoenfield. *Mathematical Logic.* Addison-Wesley, 1967.

[Smu68] Raymond Smullyan. *First-Order Logic.* Springer-Verlag, 1968.

[TM87] W.M. Turski and T.S.E. Maibaum. *The Specification of Computer Programs.* Addison-Wesley, 1987.

The Tableau Algorithm for Intuitionistic Propositional Calculus as a Constructive Completeness Proof (Extended Abstract)

Judith Underwood*

March 11, 1993

1 Introduction

Intuitionistic logic and constructive mathematics have long been known to have important connections with computer science. Much work in constructive mathematics has been concerned with the extraction of algorithms from proofs. The proof presented here was developed by the opposite approach; the constructive proof was inspired by the tableau algorithm. One interesting result of this approach is the insight provided into the connection between Kripke models and intuitionistic proofs. If the formula is unprovable, a model which does not force the formula is created from the failed tableau proof; in a sense, they have the same "data structure". If the formula is provable, evidence for it is produced in the form of a lambda term of the corresponding type. In addition, the description of the tableau algorithm which arises from the proof is easy to understand, and in particular (and unlike many other presentations) it is very easy to see that the algorithm terminates. The algorithm has double exponential time complexity; the problem is known to be Pspace-complete [8].

One purpose of this paper is to show that the completeness proof is suitable for formalization in Nuprl [1]; using Nuprl's extraction mechanism, a verified implementation of the algorithm can then be produced from the formal constructive proof. Hence, definitions will be in the notation of type theory, and the propositional logic proof of the formula (if one exists) will be constructed as an inhabitant of the type corresponding to the proposition. The mathematical and logical equivalents follow from the Curry-Howard isomorphism.

The remainder of this paper consists of three sections: a presentation of the definitions necessary for the theorem and the formal statement of the theorem, the proof of the theorem, and a conclusion. Good introductions to intuitionistic logic are Dragalin's book [3] and Nerode's notes [7]. The latter describes the tableau algorithm for this logic in detail. A nice presentation of a variant of the tableau algorithm is found in Dyckhoff [4]. Girard [6] discusses the Curry-Howard isomorphism. The proof is modelled after the completeness proof in Fitting [5]. A presentation of the tableau algorithm as a constructive proof of decidability for classical propositional logic appears in Constable and Howe [2], which also contains an introduction to Nuprl, and further discussion of the idea of implementing and analyzing theorem provers via metamathematics. An earlier version of this paper is available as a technical report [9].

2 Definitions and Statement of Theorem

The first problem is to translate the mathematical definition of validity to type theory. We choose the most general definition of Kripke model, and modify only slightly the definition of forcing.

*Supported by a National Science Foundation Graduate Fellowship and a Mathematical Sciences Institute Graduate Fellowship. Author's address: Department of Computer Science, Upson Hall, Cornell University, Ithaca, NY 14853, USA. Email: under@cs.cornell.edu.

In mathematical terms, a Kripke model is a triple consisting of a set, a transitive and reflexive relation on that set, and a monotone function from the set to the set of atomic formulas. The set can be thought of as "states of knowledge"; the order relation then describes increasing information, as the set of atomic formulas associated with each state is considered to be the atomic formulas known to be true at that state.

Although Nuprl type theory has set types, it is simpler in this case to consider the set of states of knowledge as a type itself. This is expressed by giving it the type U_1, the lowest in a hierarchy of universe types, that is, types whose members are smaller types. See [1] for more information. The type theoretic definition of Kripke model is then:

$$\begin{aligned} \mathit{Kripke\_model} \;=\; & T : U_1 \\ & \#R : \mathit{transitive\_reflexive\_relation}(T, T) \\ & \#af : \mathit{monotone\_relation}(T, \mathit{Atomic\_formulas}) \end{aligned}$$

Forcing in a particular Kripke model $K = (T, R, af)$ is inductively defined as a predicate $\mathit{forces}(K, s, \alpha)$, where $s \in T$ and $\alpha \in \mathit{Term}$. At the same time (i.e. by mutual recursion), to avoid troublesome negations in the definition of forcing, a predicate $\mathit{notforces}(K, s, \alpha)$ is also defined. As long as terms are defined inductively, it is not difficult to implement these definitions in Nuprl. The formal definition of *notforces* is omitted for brevity.

$$\begin{array}{lll} \mathit{forces}(K, s, \alpha) \;=\; & \mathit{af}(s, \alpha) & (\alpha \in \mathit{Atom}) \\ & \mathit{forces}(K, s, \beta) \wedge \mathit{forces}(K, s, \gamma) & (\alpha = \beta \wedge \gamma) \\ & \mathit{forces}(K, s, \beta) \vee \mathit{forces}(K, s, \gamma) & (\alpha = \beta \vee \gamma) \\ & \forall s'.sRs' \rightarrow & \\ & \mathit{notforces}(K, s', \beta) \vee \mathit{forces}(K, s', \gamma) & (\alpha = \beta \rightarrow \gamma) \end{array}$$

Let *Term* be the type of propositional formulas. As in [2], the proof is described in terms of *sequents.*

$$\mathit{Sequent} = (\mathit{Term\ list}) \# (\mathit{Term\ list})$$

However, more information is required for the intuitionistic case; thus our basic unit will be a system of sequents.

$$\mathit{System} = (\mathit{Sequent\ list})$$

(The list type is a convenience; sequents should be thought of as pairs of sets and a system of sequents should be considered a set of sequents.) A given sequent will be written as a pair $\langle S_T, S_F \rangle$. The formulas in S_T are the ones which are tagged "true" at some node in the tableau algorithm; similarly the formulas in S_F are tagged "false". The sequents themselves correspond to the nodes in the tableau algorithm; they will also correspond to the states in the Kripke model disproving validity, if one exists.

Although we are generally interested only in the validity of a single formula φ, or rather a given sequent $\langle \emptyset, \varphi \rangle$, the induction requires a statement which is true for systems of sequents. We will prove that, for any system, either there is a sequent in this system which is provable or there is a Kripke model which, for all sequents $\langle S_T, S_F \rangle$ in the system, forces the formulas in S_T but not those in S_F. This Kripke model will be referred to as a *countermodel* for the system.

A sequent $\langle S_T, S_F \rangle$ is provable if there is an inhabitant of the type corresponding to $\bigwedge S_T \rightarrow \bigvee S_F$. In the remainder of this abstract, we shall use propositional notation for types.

If there is a countermodel, we also supply a function f from the system to the model such that $f(\langle S_T, S_F \rangle)$ is the state in the system which forces the formulas in S_T but not those in S_F.

The formal statement of the theorem is then

$$
\begin{array}{l}
\forall \Sigma : \mathit{System}. \\
\quad (\exists \langle S'_T, S'_F \rangle \in \Sigma. (\exists f : \bigwedge S'_T \to \bigvee S'_F)) \\
\vee \\
(\exists K = (T, R, af) : \mathit{Kripke\_model}. \\
\quad \exists f : \Sigma \to T. \\
\qquad \forall \langle S_T, S_F \rangle \in \Sigma. \\
\qquad (\forall \alpha \in S_T. \mathit{forces}(K, f(\langle S_T, S_F \rangle), \alpha) \wedge \forall \alpha \in S_F. \mathit{notforces}(K, f(\langle S_T, S_F \rangle), \alpha))))
\end{array}
$$

Note that when the theorem is applied to a formula φ, Σ will contain exactly one sequent, namely $\langle \emptyset, \varphi \rangle$; thus we have either φ is provable or there is a countermodel which does not force φ.

3 Proof of the Theorem

The proof is by induction on the number of formulas which can be added to the system. Each step either adds a formula to a sequent in Σ or adds another sequent to Σ. All formulas added will be subformulas of the formulas already in Σ, hence only finitely many can be added to a particular sequent. A new sequent $\langle S_T, S_F \rangle$ is added only when no more formulas can be added to sequents already in Σ, and then only if there is no sequent $\langle S'_T, S'_F \rangle$ in Σ with $S_T \subseteq S'_T$ and $S_F \subseteq S'_F$, so that any new sequent will never develop into a sequent already present in Σ. Also, the formulas in $\langle S_T, S_F \rangle$ will be subformulas of the formulas already in Σ, so only finitely many new sequents can be added.

The proof proceeds by cases, one case for each choice of active formula. In this abstract, we present one case which is representative of most of the cases, plus two unique cases.

1. For any pair $\langle S_T, S_F \rangle \in \Sigma$, if $\alpha \wedge \beta \in S_T$ and ($\alpha \notin S_T$ or $\beta \notin S_T$), apply the inductive hypothesis to the larger system Σ', where Σ' is Σ with $\langle S_T, S_F \rangle$ replaced by $\langle S_T \cup \{\alpha, \beta\}, S_F \rangle$.

Suppose, for the system Σ', it is the case that $\alpha \wedge \beta \wedge (\bigwedge S_T) \to \bigvee S_F$, i.e. we have an inhabitant $f' = \lambda x^{\alpha \wedge \beta \wedge (\bigwedge S_T)}.t$ of the corresponding function type. Then we can produce an inhabitant f of $\bigwedge S_T \to \bigvee S_F$ as follows. Given an element a of type $\bigwedge S_T$, set f equal to $f'(\pi_1(\pi_k(a)^{\alpha \wedge \beta}), \pi_2(\pi_k(a)^{\alpha \wedge \beta}), a)$ (where k is the index of $\alpha \wedge \beta$ in S_T). Then, since $\alpha \wedge \beta \in S_T$, $f \in \bigwedge S_T \to \bigvee S_F$. (If some other sequent in Σ' is the one corresponding to the inhabited type, since it is also in Σ the theorem still holds.)

If we have a countermodel $K = (T, R, af)$ for Σ', and we have $f : \Sigma' \to T$ such that $f(\langle S_T \cup \{\alpha, \beta\}, S_F \rangle) = s$ for some $s \in T$ (so that $\forall \gamma \in S_T \cup \{\alpha, \beta\}. \mathit{forces}(K, s, \gamma)$), then we trivially have $\forall \gamma \in S_T. \mathit{forces}(K, s, \gamma)$. Letting $f(\langle S_T, S_F \rangle) = s$, we then have that (T, R, af) is a Kripke countermodel for Σ.

Most of the other cases proceed as this one. The exception is the following case, where an implication which is tagged false is chosen to be reduced. We choose not to reduce these until all other types of formulas have been reduced.

2. For any pair $\langle S_T, S_F \rangle \in \Sigma$, if $\alpha \to \beta \in S_F$ and $\forall \langle S'_T, S'_F \rangle \in \Sigma. (S_T \cup \{\alpha\} \not\subseteq S'_T \vee \beta \notin S'_F)$, and none of the other cases hold, apply the inductive hypothesis to Σ', where Σ' is $\Sigma \cup \{\langle S_T \cup \{\alpha\}, \{\beta\} \rangle\}$. Note $\langle S_T, S_F \rangle$ is in Σ'.

If, for the system Σ', we have an inhabitant f' of $(\bigwedge S_T \wedge \alpha) \to \beta$, then the function $\lambda y^{\wedge S_T}. \lambda x^{\alpha}. f'(y, x)$ is in $\bigwedge S_T \to (\alpha \to \beta)$, so the appropriate tagged version of this function is in $\bigwedge S_T \to \bigvee S_F$.

If we have a countermodel $K = (T, R, af)$ for Σ' and the function $f : \Sigma' \to T$, then let $s = f(\langle S_T, S_F \rangle)$. Then we immediately have that $\forall \gamma \in S_T. \mathit{forces}(K, s, \gamma)$ and $\forall \gamma \in S_F. \mathit{notforces}(K, s, \gamma)$. So, we need not change f for Σ, except to remove $\langle S_T \cup \{\alpha\}, \{\beta\} \rangle$ from its domain. We then have that (T, R, af) is a Kripke countermodel for Σ.

3. If no formula can be reduced as described above, we have reached the base case. If there is an $\langle S_T, S_F\rangle \in \Sigma$ such that $\exists \alpha \in S_T \cap S_F$, then $\bigwedge S_T \rightarrow \bigvee S_F$ is inhabited by $\lambda x^{\wedge S_T}.(j, \pi_i(x))$, where i is the index of α in $\bigwedge S_T$ and j is the tag for α in the disjoint union $\bigvee S_F$. Also, if $\bot$ is in some S_T, then $\bigwedge S_T \rightarrow \bigvee S_F$ is inhabited by $\lambda x^{\wedge S_T}.any(\pi_i(x))$, where i is the index of $\bot$ in S_T.

Otherwise, $\forall \langle S_T, S_F\rangle \in \Sigma$, $S_T \cap S_F = \emptyset$. Let $T = \Sigma$, and define R, a relation on $T\#T$, and af, a relation on $T\#Atomic\_formulas$, by

$$\begin{aligned} R(\langle S_T, S_F\rangle, \langle S'_T, S'_F\rangle) &\Leftrightarrow S_T \subseteq S'_T \\ af(\langle S_T, S_F\rangle, \alpha) &\Leftrightarrow \alpha \in S_T. \end{aligned}$$

It is easy to see that R is transitive and reflexive, and that af is monotone with respect to R, so $(T, R, af) : Kripke\_model$. Note that since $S_T \cap S_F = \emptyset, \alpha \in S_F \rightarrow \neg af(s, \alpha)$.

Define $f : \Sigma \rightarrow T$ to be the identity map from Σ to T. To complete the theorem, we must show that

$$\begin{aligned} &\forall \langle S_T, S_F\rangle \in \Sigma. \\ &\quad (\forall \alpha \in S_T.forces(K, f(\langle S_T, S_F\rangle), \alpha) \wedge \forall \alpha \in S_F.notforces(K, f(\langle S_T, S_F\rangle), \alpha)). \end{aligned}$$

Since *forces* and *notforces* are defined by mutual induction on the complexity of the formula, the proofs of $(\forall \alpha \in S_T.forces(K, s, \alpha) \wedge \forall \alpha \in S_F.notforces(K, s, \alpha))$ must proceed similarly. This is easily done by induction on the complexity of α.

4 Conclusion

Each system of sequents in the above proof corresponds to a branch of the tableau. If all branches close (i.e. if there is a sequent in each system such that $S_T \cap S_F$ is nonempty), then the formula is provable; else, the open branch becomes a Kripke model which does not force the formula. Note that a state is added to the Kripke model only when necessary to provide a state which fails to force an implication; hence, the model produced is minimal (when states which force the same formulas are identified). The model produced is finite, so the proof also demonstrates the completeness of finite Kripke models.

If the proof were added to the Nuprl system, the result would be a procedure for deciding the (intuitionistic) validity of a propositional formula via the tableau algorithm. The author and others at Cornell are currently working on describing this proof in such a way that Nuprl's reflection mechanism can use the computational content to reason about Nuprl terms, and not just abstract propositions. This will demonstrate how Nuprl can extend its own store of theorem proving techniques in a verified and sound way.

References

[1] R. L. Constable *et al. Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[2] R. L. Constable and D. J. Howe. Implementing Metamathematics as an Approach to Automatic Theorem Proving. In *Formal Techniques in Artificial Intelligence*, R. Banerji, ed. North-Holland, 1990.

[3] A. G. Dragalin. *Mathematical Intuitionism: Introduction to Proof Theory,* American Mathematical Society, Translations of Mathematical Monographs, Volume 67, 1988.

[4] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic, Journal of Symbolic Logic 57 (1992), 795-807.

[5] M. Fitting. *Intuitionistic Logic, Model Theory, and Forcing.* North-Holland, 1969.

[6] J. Y. Girard. *Proofs and Types.* Cambridge University Press, 1989.

[7] A. Nerode. Some Lectures on Intuitionistic Logic. In *Logic and Computer Science*, Springer-Verlag Lecture Notes in Mathematics 1429, 1990.

[8] R. Statman. Intuitionistic Propositional Logic is Polynomial-space Complete. Theoretical Computer Science 9, 67-72, 1979.

[9] J. Underwood. A constructive completeness proof for the intuitionistic propositional calculus. Technical report 90-1179, Cornell University, 1990.

CLAUSAL TABLEAUX WITH LINKS AND LEMMAS
Abstract

Graham Wrightson
Department of Computer Science
The University of Newcastle
NSW 2308
AUSTRALIA
email: graham@cs.newcastle.edu.au

Systematic tableaux (a la Smullyan) for clauses are complete. However, if we introduce links to systematic tableaux, in a fashion similar to the use of links in connection graphs, then we discover that these tableaux are not complete. By modifying the beta-rule and performing depth-first control we can develop tableaux which, in a way yet to be explained in the full paper, resemble resolution. Now these resolution-like clausal tableaux with links are complete if multi-tableaux (also to be described in the full paper) are used. This is because of the use of lemmas which are generated in a resolution-like way whilst taking into account the creation of Hintikka sets to guarantee the completeness of the resolution-like tableaux.

What is interesting in all of this is that the deletion property of the links in the resolution-like tableaux, just as in connection graphs, forces the use of lemmas.