Computing Stable Models by Program Transformation

Jürgen Stuber

MPI–I–93–257 December 1993

Author's Address

Jürgen Stuber, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, `juergen@mpi-sb.mpg.de`

Acknowledgements

The research described in this report was supported by the German Ministry for Research and Technology (Bundesministerium für Forschung und Technologie) under grant ITS 9103.

Abstract

In analogy to the Davis-Putnam procedure we develop a new procedure for computing stable models of propositional normal disjunctive logic programs, using case analysis and simplification. Our procedure enumerates all stable models without repetition and without the need for a minimality check. Since it is not necessary to store the set of stable models explicitly, the procedure runs in polynomial space.

We allow clauses with empty heads, in order to represent truth or falsity of a proposition as a one-literal clause. In particular, a clause of form $\sim A \rightarrow$ expresses that A is constrained to be true, without providing a justification for A. Adding this clause to a program restricts its stable models to those containing A, without introducing new stable models. Together with $A \rightarrow$ this provides the basis for case analysis.

We present our procedure as a set of rules which transform a program into a set of solved forms, which resembles the standard method for presenting unification algorithms. Rules are sound in the sense that they preserve the set of stable models. A subset of the rules is shown to be complete in the sense that for each stable model a solved form can be obtained. The method allows for concise presentation, flexible choice of a control strategy and simple correctness proofs.

Keywords

Stable semantics, stable models, normal disjunctive logic program, normal deductive database, default negation, negation-as-failure, Davis-Putnam.

1 Introduction

Stable models have been introduced by Gelfond and Lifschitz (1988) as a semantics for logic programs. Most methods build stable models bottom-up, using clauses in the forward direction. For clauses whose premises are satisfied in a partially built interpretation, one proposition of the head is made true, which requires a case split for disjunctive clauses. The propositions of the negative premises are recorded to be false, either by collecting them in a separate set or by adding special literals to the interpretation. An interpretation is discarded if an inconsistency is detected. Saccà and Zaniolo (1990) give a procedure which finds a single stable model of a normal non-disjunctive program. Nevertheless it can easily be modified to allow enumerating all stable models. However, because clauses with independent assumptions may be considered in any order, a stable model may be generated many times. Fernández and Minker (1992) and Inoue, Koshimura and Hasegawa (1992) compute stable models of normal disjunctive programs. To cope with disjunction they explicitly store sets of models in order to test for minimality, which may require exponential space in the worst case. Eshghi (1990) uses the Assumption-based Truth Maintenance System (ATMS) for computing stable models of non-disjunctive programs. It starts with inconsistent assumptions and relaxes them until stable models are reached. Its representation of minimal sets of inconsistent assumptions (nogoods) and minimal sets of assumptions needed to derive a certain proposition (environments) may also require exponential space in the worst case.

We present a new procedure which computes the stable models of a normal disjunctive logic program (a normal deductive database). For infinite Herbrand bases the problem is known to be infeasible in general. There may exists continuum many stable models, and the problem of deciding whether a recursively enumerable interpretation is a stable model is Π^0_2-hard (Marek and Subrahmanian 1992). Therefore we limit ourselves to the finite case, like all other currently known methods for computing stable models. This still includes the practically important case of datalog programs.

Our procedure uses case analysis with respect to the truth-value of propositions, in analogy to Davis and Putnam (1960). This enables it to enumerate the stable models for finite propositional normal disjunctive logic programs without repetitions and without the need to explicitly test for minimality. Rather than testing afterwards, we take care to preserve minimality during computation. Each model is generated only once because cases do not overlap.

Eiter and Gottlob (1993) have shown that the problem of deciding whether a stable model exists is Σ_2-complete in the polynomial hierarchy. Our procedure reflects this result; it uses nondeterministic polynomial time, and the disjunction rule uses *co-SAT* from *co-NP* as an oracle.

Our procedure tries to transform a normal disjunctive logic program into sets of clauses in solved form which for each proposition contain exactly one of the clauses $A \rightarrow$ or $\rightarrow A$.[1] This makes determining its stable model trivial. One-literal clauses play a special role in our procedure, since they are used to represent truth or falsity of a proposition, as specified by the following table.

$A \rightarrow$	A is false
$\rightarrow A$	A is true and justified
$\sim A \rightarrow$	A is true but not justified

A true but unjustified proposition acts as a constraint; it does not help in the construction

[1]Note that we allow clauses with empty heads.

of a stable model, but is used to check the truth of A after the model has been constructed. Operationally, if $A \rightarrow$ is ever added to a program containing $\sim A \rightarrow$, an inconsistency results. $A \rightarrow$ or $\rightarrow A$ are used to simplify the program, removing all other occurrences of A. With $\sim A \rightarrow$ we may remove clauses containing $\sim A$—these are subsumed—and we may reduce positive premises A from clauses with an empty head. Other simplifications are not possible because we have to be careful not to add unwarranted justification for A. Another use of $\sim A \rightarrow$ is to block case analysis on A, since then its truth-value is already known. The basis for case analysis is the observation that adding alternatively $A \rightarrow$ or $\sim A \rightarrow$ to a program P splits its set of stable models exactly into the two sets of stable models of the new programs. One contains all stable models of P where A is false while the other contains those where A is true.

Case analysis and simplification are sufficient to compute all stable models of non-disjunctive programs. For disjunctive programs only a very restricted case remains open, where the constraints specify exactly one possible stable model. It suffices to test whether the disjunctions provide justification for all propositions constrained to true. To this end it is necessary to prove the validity of certain classical implications. Any procedure for tautology checking in classical propositional logic may be employed, for instance classical Davis-Putnam.

We prove the soundness of some other rules which are not needed for completeness but which may speed up the computation, namely Tautology elimination, Factoring and Default negation. Default negation corresponds to the Purity rule of the classical Davis-Putnam procedure for the case of propositions which don't appear positively. It is our only nonmonotonic rule.

For programs without a stable model Inoue et al. (1992) distinguish between *inconsistent* and *incoherent* programs. Incoherent programs are those which do not allow to derive an inconsistency, but which have no stable model for lack of justification. Suppose we do not use the Default negation rule. Then an incoherent program is transformed into at least one program which is reduced with respect to our rules, not solved and does not contain the empty clause. The distinction between solved forms and reduced incoherent programs will then be the only source of nonmonotonicity. If incremental computation is desired, one may store these reduced incoherent programs and resume the computation when new clauses are added.

We present our procedure as a set of rules which transform a program into a set of solved forms, which resembles the standard method for presenting unification algorithms. Rules are sound in the sense that they preserve the set of stable models. Subsets of the rules are shown to be complete for non-disjunctive and disjunctive programs, in the sense that for each stable model a solved form can be obtained. This presentation has several advantages. Soundness, termination and completeness can be proven with relative ease. It is made precise which transformations are necessary to achieve completeness, while more sound transformation may be used where appropriate. Any inference system between these boundaries is sound and complete. We also make precise which choices can be made eagerly, and where alternative cases have to be considered.

2 Preliminaries

$\uplus$ denotes disjoint union. We assume a fixed set of propositions *Prop*. A (normal disjunctive) clause C is a triple $\langle \mathit{Prem}^{+}(C), \mathit{Prem}^{\sim}(C), \mathit{Concl}(C) \rangle$ of sets of propositions $\mathit{Prem}^{+}(C) = \{A_1, \ldots, A_n\}$, $\mathit{Prem}^{\sim}(C) = \{B_1, \ldots, B_m\}$ and $\mathit{Concl}(C) = \{D_1, \ldots, D_k\}$, which is written

$$A_1, \ldots, A_n, \sim B_1, \ldots, \sim B_m \rightarrow D_1, \ldots, D_k.$$

Given $\Gamma = A_1, \ldots, A_n$ we write $\sim\cdot\Gamma$ for $\sim A_1, \ldots, \sim A_n$. A clause C is *non-disjunctive* if $k \leq 1$ and *negation-free* if $m = 0$. A (normal disjunctive logic) program is a set of clauses. A program is *non-disjunctive* if all its clauses are non-disjunctive. We write *Prog* for the set of programs, and *NProg* for the set of non-disjunctive programs.

An *interpretation* I is a set of propositions. I *satisfies* a negation-free clause C, written $I \models C$, iff $\mathit{Prem}^+(C) \not\subseteq I$ or $\mathit{Concl}(C) \cap I \neq \emptyset$. I *satisfies* a negation-free program P, written $I \models P$, iff it satisfies all clauses in P. In this case we also say that I is a *model* of P. The set of models of P is denoted by $\mathit{Mod}(P)$. P' is a *logical consequence* of P, written $P \models P'$, if $\mathit{Mod}(P) \subseteq \mathit{Mod}(P')$. An $I \in \mathit{Mod}(P)$ is *minimal* if there exists no $J \in \mathit{Mod}(P)$ such that J is a proper subset of I. $\mathit{Min}(P)$ denotes the set of minimal models of a program P. For a program P let $\tilde{P}$ be the program where each literal $\sim A$ is replaced by a new proposition $\tilde{A}$. Analogously define $\tilde{C}$ for a clause C. Let $\widetilde{\mathit{Prop}} = \{\, \tilde{A} \mid A \in \mathit{Prop} \,\}$.

Gelfond and Lifschitz (1988) define a transformation on logic programs which eliminates all literals $\sim A$ from a program P based on an interpretation I, where $\sim A$ is treated as true if A is not in I. For an interpretation I, a clause C and a program P let

$$\mathit{GL}_I(C) = \begin{cases} \{\mathit{Prem}^+(C) \to \mathit{Concl}(C)\} & \text{if } \mathit{Prem}^\sim(C) \cap I = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathit{GL}_I(P) = \bigcup_{C \in P} \mathit{GL}_I(C)$$

If $\mathit{GL}_I(C) = \emptyset$ we say GL_I *eliminates* C. $\mathit{GL}_I(P)$ is called the *Gelfond-Lifschitz transformation* of P with respect to I. An interpretation I is called a *stable model* of P if I is a minimal model of $\mathit{GL}_I(P)$. We write $\mathit{Stab}(P)$ for the set of stable models of P. If we restrict this to the case of non-disjunctive clauses we get the original notion of stable model as defined by Gelfond and Lifschitz (1988).

3 Transformation rules

To describe our procedure we will use transformation rules on programs. They allow to take whole program into account, which is necessary to make nonmonotonic inferences. A program will be transformed into a set of solved forms, which correspond to the programs stable models.

A program P is *in solved form* if it consists of clauses of form $A \to$ or $\to A$ with exactly one clause for each proposition A. For a program P in solved form we define the corresponding interpretation I_P as $\{\, A \mid \to A \in P \,\}$. On the other hand an interpretation I corresponds to the program in solved form $P_I = \{\, \to A \mid A \in I \,\} \cup \{\, A \to \mid A \in \mathit{Prop} - I \,\}$.

Lemma 3.1 *Let P be a program. If P is in solved form then $\mathit{Stab}(P) = \{I_P\}$.*

A *transformation relation* is a subset $\vdash$ of *Prog* $\times$ *Prog*. We write $\vdash^*$ for its reflexive-transitive closure. A transformation rule r is a set of instances $\rho = \frac{P}{P_1|\ldots|P_n}$. An instance ρ defines a transformation relation $\vdash_\rho$ where $P \vdash_\rho P'$ if $\rho = \frac{P}{\ldots|P'|\ldots}$. To a rule r the associated transformation relation is $\vdash_r = \bigcup_{\rho \in r} \vdash_\rho$. Similarly, for a set of rules R we define $\vdash_R = \bigcup_{r \in R} \vdash_r$. We also say ρ is an instance of R.[2] We say P is *R-reduced* if there exists no P' such that $P \vdash_R P'$.

[2]We do this two-step construction, since we want to keep track which instances belong to the same rule, without specifying formally how to instantiate the schemas defining rules.

A *strategy* $\vdash_S$ for R is a subrelation of $\vdash_R$ such that if P is not R-reduced then $(\vdash_\rho) \subseteq (\vdash_S)$ for some instance ρ of R. I.e., it is sufficient to eagerly apply some arbitrarily chosen rule. Rules with more than one conclusion encode the control information that several alternatives must be considered, either by backtracking or in parallel.

Let $\vdash$ be a transformation relation, R a set of transformation rules and $\mathcal{P}$ a class of logic programs. $\vdash$ is *sound* if $P \vdash P'$ implies $\mathit{Stab}(P) = \bigcup_{P \vdash P'} \mathit{Stab}(P')$. $\vdash$ is complete with respect to $\mathcal{P}$ if $\mathit{Stab}(P) = \{\, I_{P'} \mid P \vdash^* P' \text{ and } P' \text{ is in solved form} \,\}$ for all P in $\mathcal{P}$. $\vdash$ is *terminating* if there exists no infinite sequence $P_0 \vdash P_1 \vdash P_2 \vdash \ldots$. $\vdash$ *preserves* $\mathcal{P}$ if $P \in \mathcal{P}$ and $P \vdash P'$ implies $P' \in \mathcal{P}$. R preserves $\mathcal{P}$ if $\vdash_R$ preserves $\mathcal{P}$. R is *sound*, *terminating* or *complete* if every strategy for R is sound, terminating or complete, respectively. A set of transformation rules R is *independent* with respect to $\mathcal{P}$ if there is no proper subset R' of R such that R' is complete with respect to $\mathcal{P}$.

Lemma 3.2 *Let R be a set of transformation rules.*

1. *If $\vdash_r$ is sound for all r in R then $\vdash_R$ is sound.*
2. *R is sound if and only if $\vdash_\rho$ is sound for all instances ρ of R.*

Proof: (1) If $\vdash_r$ is sound for all r in R then $\mathit{Stab}(P) = \bigcup_{P \vdash_r P'} \mathit{Stab}(P')$ for all r in R, and thus $\bigcup_{P \vdash_R P'} \mathit{Stab}(P') = \bigcup_{r \in R} \bigcup_{P \vdash_r P'} \mathit{Stab}(P') = \bigcup_{r \in R} \mathit{Stab}(P) = \mathit{Stab}(P)$.

(2) Suppose $\vdash_r$ is sound for all r in R, then $\vdash_R$ is sound by (1). By definition $P \vdash_S P'$ implies $P \vdash_R P'$. Hence $\bigcup_{P \vdash_S P'} \mathit{Stab}(P') \subseteq \bigcup_{P \vdash_R P'} \mathit{Stab}(P') = \mathit{Stab}(P)$. On the other hand, $\{\, P' \mid P \vdash_\rho P' \,\} \subseteq \{\, P' \mid P' \vdash_S P' \,\}$ for some instance ρ of R, and thus $\mathit{Stab}(P) = \bigcup_{P \vdash_\rho P'} \mathit{Stab}(P') \subseteq \bigcup_{P \vdash_S P'} \mathit{Stab}(P')$.

For the converse, assume that R is sound, but $\vdash_\rho$ is not sound for some instance $\rho = \frac{P}{P_1 | \ldots | P_n}$ of R. Then $\mathit{Stab}(P) \neq \bigcup_{1 \le i \le n} \mathit{Stab}(P_i)$. We may choose $\vdash_S$ such that $P \vdash_S P'$ iff $P \vdash_r P'$, which makes $\vdash_S$ unsound. □

4 Sound program transformations

We first state the soundness of some general program transformation rules. Later we will use this to show soundness of the specific transformation rules of our procedure for computing stable models.

We call clauses of form $A, \Gamma \to A, \Delta$ *standard tautologies* and clauses of form $A, \sim A, \Gamma \to \Delta$ *constraint tautologies*. Tautologies are always satisfied and do not contribute to the meaning of a program. We may remove tautologies.

Tautology elimination $\quad \dfrac{\{C\} \uplus P}{P} \quad$ if C is a tautology.

We have two different inference rules for resolution. We may resolve a negative literal either with a positive or a constraint literal. In the latter case a first approach might be the following inference rule.

Naïve constraint resolution $\quad \dfrac{\sim A, \Gamma \to \Delta \qquad A, \Lambda \to \Pi}{\Gamma, \Lambda \to \Delta, \Pi}$

But with this rule it is possible to create new justification for the propositions in Π. The simplest example is the program $P = \{\sim p \rightarrow; p \rightarrow p\}$, which has no stable model. Naïve constraint resolution would add the clause $\rightarrow p$, which leads to the stable model $\{p\}$. Therefore we will adopt the following inference rules for resolution.

Standard resolution $$\frac{\Gamma \rightarrow \Delta, A \qquad A, \Lambda \rightarrow \Pi}{\Gamma, \Lambda \rightarrow \Delta, \Pi}$$

Constraint resolution $$\frac{\sim A, \Gamma \rightarrow \Delta \qquad A, \Lambda \rightarrow \Pi}{\Gamma, \Lambda, \sim \cdot \Pi \rightarrow \Delta}$$

The conclusion of such a resolution inference is called a *resolvent* of the premises. We may add resolvents.

Resolution $$\frac{P}{\{C\} \cup P} \quad \text{if } C \text{ is a resolvent of two clauses in } P.$$

Subsumption is useful to remove clauses which are already implied by the program, in order to reduce the size of the program. Moreover, we will use it to explicitly remove clauses which are satisfied. Note the interaction between positive and constraint literals. In particular, $\rightarrow A$ subsumes $\sim A \rightarrow$. A clause C' *subsumes* a clause C if $\mathit{Prem}^+(C') \subseteq \mathit{Prem}^+(C)$, $\mathit{Prem}^\sim(C') \subseteq \mathit{Prem}^\sim(C)$ and $\mathit{Concl}(C') \subseteq \mathit{Concl}(C) \cup \mathit{Prem}^\sim(C)$. We may remove subsumed clauses.

Subsumption $$\frac{\{C, C'\} \uplus P}{\{C'\} \uplus P} \quad \text{if } C' \text{ subsumes } C.$$

A clause $\sim A, \Gamma \rightarrow \Delta$ is called a *factor* of $\sim A, \Gamma \rightarrow A, \Delta$. Note that a proposition cannot appear twice in the same role in a clause, since for each clause the positive premises, the negative premises and the conclusions form sets. Hence we do not need other factoring rules.

Factoring $$\frac{P}{\{C\} \cup P} \quad \text{if } C \text{ is a factor of a clause in } P.$$

We can explain the effect of a clause with negative loop like $\sim p \rightarrow p$ by observing that it can be factored to $\sim p \rightarrow$. Hence it constrains p to be true.
The classical Davis-Putnam procedure has a Purity-rule, which stipulates a propositions to be true if it only appears in positive positions and false if it only appears in negative positions. Since we need to preserve minimality, we can only do the latter here. In fact, if a proposition does not appear in any head of a clause it cannot be true in a stable model.

Default negation $$\frac{P}{\{A \rightarrow\} \cup P} \quad \text{if no head of a clause in } P \text{ contains } A.$$

In contrast to the other rules in this section, Split is not an equivalence transformation on programs, but generates two programs whose combined semantics is equivalent to the original. The two cases correspond to A being either true or false.

Split $$\frac{P}{\{A \rightarrow\} \uplus P \quad | \quad \{\sim A \rightarrow\} \uplus P}$$

Unlike in the classical Davis-Putnam procedure, case analysis is not sufficient to cope with disjunctive programs in the case of stable semantics. We can remove all false disjuncts by resolution with the negative fact $A \rightarrow$, but if in a minimal model more than one disjunct is true the constraint $\sim A \rightarrow$ is too weak to remove the disjunction.

Disjunction $$\frac{\{\rightarrow A_1, \ldots, A_n\} \uplus P}{\{\rightarrow A_1; \ldots; \rightarrow A_n; \rightarrow A_1, \ldots, A_n\} \cup P}$$

if $n \geq 2$ and $\tilde{P} \cup \{\rightarrow A_1, \ldots, A_n\} \models \{\rightarrow A_1; \ldots; \rightarrow A_n\}$.

Disjunction factors out the problem, enabling us to apply any suitable method to it. For instance, we may check if $\tilde{P} \cup \{\rightarrow A_1, \ldots, A_n\} \cup \{A_1, \ldots, A_n \rightarrow\}$ is inconsistent by using a classical Davis-Putnam procedure.

Theorem 4.1 *Tautology elimination, Resolution, Subsumption, Factoring, Default negation, Split and Disjunction are sound transformation rules.*

For the proof of this theorem we refer the reader to the appendix.

5 The Davis-Putnam procedure

Factoring, Disjunction and certain Resolution transformations add clauses which subsume a clause in their premise. We may remove this clause by subsumption. In combination this yields simplification rules which decrease the size of the program. Moreover, we will also use Tautology elimination and Subsumption for simplification, and in particular subsumption with respect to one-literal clauses, as this can be implemented most efficiently.

Split, Disjunction, Default negation and Constraint propagation add one-literal clauses, thereby decreasing the number of propositions whose truth value is unknown. When proving termination, we will give this decrease precedence over the increase in size. Also, the increase in size is at most linear in the number of propositions. We call a proposition A *unknown* in P if it doesn't appear in a one-literal clause in P.

Subsumption-true (SubT) $$\frac{\{\Gamma \rightarrow \Delta, A; \rightarrow A\} \uplus P}{\{\rightarrow A\} \uplus P}$$

Subsumption-false (SubF) $$\frac{\{\Gamma, A \rightarrow \Delta; A \rightarrow\} \uplus P}{\{A \rightarrow\} \uplus P}$$

Subsumption-constraint (SubC) $$\frac{\{\Gamma, \sim A \rightarrow \Delta; \sim A \rightarrow\} \uplus P}{\{\sim A \rightarrow\} \uplus P}$$

GL-Subsumption (SubGL) $$\frac{\{\Gamma, \sim A \rightarrow \Delta; \rightarrow A\} \uplus P}{\{\rightarrow A\} \uplus P}$$

Reduction-true (RedT) $$\frac{\{\Gamma, A \rightarrow \Delta; \rightarrow A\} \uplus P}{\{\Gamma \rightarrow \Delta\} \cup \{\rightarrow A\} \cup P}$$

Reduction-false (RedF) $$\frac{\{\Gamma \rightarrow \Delta, A; A \rightarrow\} \uplus P}{\{\Gamma \rightarrow \Delta\} \cup \{A \rightarrow\} \cup P}$$

Reduction-constraint (RedC) $$\frac{\{\Gamma, A \rightarrow; \sim A \rightarrow\} \uplus P}{\{\Gamma \rightarrow\} \cup \{\sim A \rightarrow\} \cup P}$$

GL-Reduction (RedGL) $$\frac{\{\Gamma, \sim A \rightarrow \Delta; A \rightarrow\} \uplus P}{\{\Gamma \rightarrow \Delta\} \cup \{A \rightarrow\} \cup P}$$

Split (Split) $$\frac{P}{\{A \rightarrow\} \uplus P \quad | \quad \{\sim A \rightarrow\} \uplus P}$$

if A is unknown in P.

Disjunction (Disj) $$\frac{\{\rightarrow A_1, \ldots, A_n\} \uplus P}{\{\rightarrow A_1; \ldots; \rightarrow A_n\} \cup P}$$

if $n \geq 2$, for some $1 \leq i \leq n$ neither $A_i \rightarrow$ nor $\rightarrow A_i$ is in P and $\tilde{P} \cup \{\rightarrow A_1, \ldots, A_n\} \models \{\rightarrow A_1; \ldots; \rightarrow A_n\}$.

Constraint propagation (CP) $$\frac{\{A_1, \ldots, A_n \rightarrow B; \sim A_1 \rightarrow; \ldots; \sim A_n \rightarrow\} \uplus P}{\{\sim B \rightarrow; A_1, \ldots, A_n \rightarrow B; \sim A_1 \rightarrow; \ldots; \sim A_n \rightarrow\} \uplus P}$$

if B is unknown in P.

Default negation (DN) $$\frac{P}{\{A \rightarrow\} \uplus P}$$

if P contains no clause of form $\Gamma \rightarrow \Delta, A$ and $A \rightarrow$ is not already in P.

Tautology elimination (TE) $$\frac{\{C\} \uplus P}{P}$$

if C is a tautology.

Factoring (F) $$\frac{\{\sim A, \Gamma \rightarrow A, \Delta\} \uplus P}{\{\sim A, \Gamma \rightarrow \Delta\} \cup P}$$

Let *Full* be the set of all transformation rules defined in this section. Furthermore we define the following particular subsets of *Full*:

$$\begin{aligned} \mathit{Norm} &= \{\mathrm{SubF}, \mathrm{SubGL}, \mathrm{RedT}, \mathrm{RedGL}, \mathrm{Split}\} \\ \mathit{Disj} &= \mathit{Norm} \cup \{\mathrm{RedF}, \mathrm{Disj}\} \end{aligned}$$

We will show that these are complete and independent for their respective classes of programs. Note that the Gelfond-Lifschitz transformation essentially uses the transformation rules $GL = \{\mathrm{SubGL}, \mathrm{RedGL}\}$, i.e., $P \cup P_I \vdash^*_{GL} GL_I(P) \cup P_I$.

Theorem 5.1 *Every subset of Full is sound.*

Proof: It is sufficient to show $Stab(P) = \bigcup_{1\leq i\leq n} Stab(P_i)$ for every rule instance $\frac{P}{P_1|\ldots|P_n}$ of *Full*. The subsumption rules are instances of the general subsumption rule. The reduction rules can be obtained from Resolution followed by Subsumption. Disjunction and Factoring can be obtained from the general rules followed by Subsumption. Constraint propagation is an instance of Resolution. Default negation, Tautology elimination and Split are the same as the general rules. Thus soundness follows from theorem 4.1. □

Theorem 5.2 *Every subset R of Full is terminating. Moreover, the length of a derivation $P \vdash \ldots \vdash P'$ is polynomial in the size of P for any $(\vdash) \subseteq (\vdash_{Full})$.*

Proof: To each program we associate a complexity measure $c(P) = 2U(P) + 2U'(P) + |P|$ where $U(P)$ is the set of unknown propositions in P, $U'(P)$ is the set of propositions A such that P neither contains $A \to$ nor $\to A$ and $|P|$ is the number of literal occurrences in P. All subsumption and reduction rules, Tautology elimination and Factoring don't increase $2U(P) + 2U'(P)$ and strictly decrease $|P|$. Disjunction, Default negation, Split and Constraint propagation strictly decrease $2U(P) + 2U'(P)$ by at least two, while Disjunction doesn't increase $|P|$ and Default negation, Split and Constraint propagation increase $|P|$ by one. Thus every rule strictly decreases $c(P)$ and every derivation in $\vdash_{Full}$ starting from P can have at most length $c(P)$. Since any strategy for a subset of *Full* is a subset of $\vdash_{Full}$ we conclude that every subset of *Full* is terminating. □

Our transformation rules allow a great variety of strategies, but only a few will be reasonable in a practical sense. Tautology elimination and Factoring need to be used only at the beginning, since no rule introduces a tautology or a factorable clause. The simplification rules, Constraint propagation and Default negation should be used exhaustively before applying Split. For the selection of the literal to use for splitting, heuristics known from classical Davis-Putnam procedures, like selecting propositions from short clauses, should be useful. We conjecture that in practice Disjunction will be applicable rarely. Since it is expensive to test its condition, it is reasonable to try it only if no other rule is applicable. In this case for a disjunction $\to A_1, \ldots, A_n$ there exist clauses $\sim A_1 \to; \ldots; \sim A_n \to$ and a stable model can only be reached if all A_i become true. We may abort a branch of our computation once the empty clause signals an inconsistency. For instance, to determine the stable models of $P = \{\to p, q; \sim p \to p; p, \sim q \to r\}$ the following would be a typical computation.

$$
\text{Split}\;\dfrac{\text{F}\;\dfrac{\{\to p,q;\sim p\to p;p,\sim q\to r\}}{\{\to p,q;\sim p\to;p,\sim q\to r\}}}{\begin{array}[t]{c}\text{RedT}\;\dfrac{\text{RedGL}\;\dfrac{\text{SubGL}\;\dfrac{\text{RedF}\;\dfrac{\{q\to;\to p,q;\sim p\to;p,\sim q\to r\}}{\{q\to;\to p;\sim p\to;p,\sim q\to r\}}}{\{q\to;\to p;p,\sim q\to r\}}}{\{q\to;\to p;p\to r\}}}{\{q\to;\to p;\to r\}}\\[1em] \text{solved, } I_P=\{p,r\}\end{array}\;\Bigg|\;\begin{array}[t]{c}\dfrac{\dfrac{\{\sim q\to;\to p,q;\sim p\to;p,\sim q\to r\}}{\{\sim q\to;\to p,q;\sim p\to\}}\;\text{SubC}}{\{r\to;\sim q\to;\to p,q;\sim p\to\}}\;\text{DN}\\[1em] \text{incoherent, no model}\end{array}}
$$

We get the result $Stab(P) = \{\{p, r\}\}$.

The other procedures for computing stable models of disjunctive programs do a case analysis with respect to which proposition of the conclusion becomes true. As a consequence some implied proposition may also become true, and the resulting model need not be minimal. Hence they need an explicit minimality check. We do case analysis whether an atom is false or constrained to true. With a false proposition we can reduce a disjunction while preserving minimality. However, we cannot reduce a disjunction with respect to a clause $\sim A \to$, hence we get a residual case where all atoms in the disjunction are constrained to be true. The Disjunction rule is then used to handle this special situation. The following derivations illustrates what happens for two critical example programs. First consider $P = \{\to p, q; p \to q\}$.

$$\text{Split}\;\dfrac{\{\to p,q;p\to q\}}{\text{SubF}\;\dfrac{\text{RedF}\;\dfrac{\{p\to;\to p,q;p\to q\}}{\{p\to;\to q;p\to q\}}}{\{p\to;\to q\}} \;\Big|\; \dfrac{\{\sim p\to;\to p,q;p\to q\}}{\{\sim q\to;\sim p\to;\to p,q;p\to q\}}\;\text{CP}}$$

solved, $I_P = \{p\}$ incoherent, no model

In the right branch Disjunction is not applicable, since p is not a consequence of P. By adding the clause $q \to p$ we get an example where Disjunction is needed. Note that this computation is an extension of the previous one, which illustrates the possibility of incremental computation.

$$\text{Split}\;\dfrac{\{\to p,q;p\to q;q\to p\}}{\text{RedT}\;\dfrac{\text{RedT}\;\dfrac{\text{RedF}\;\dfrac{\{p\to;\to p,q;p\to q;q\to p\}}{\{p\to;\to q;p\to q;q\to p\}}}{\{p\to;\to q;p\to q;\to p\}}}{\{\Box;\to q;p\to q;\to p\}} \;\Big|\; \dfrac{\dfrac{\dfrac{\{\sim p\to;\to p,q;p\to q;q\to p\}}{\{\sim q\to;\sim p\to;\to p,q;p\to q;q\to p\}}\;\text{CP}}{\{\sim q\to;\sim p\to;\to p;\to q;p\to q;q\to p\}}\;\text{Disj}}{\dfrac{\vdots}{\{\to p;\to q\}}}}$$

inconsistent, no model

solved, $I_P = \{p, q\}$

5.1 Completeness

Lemma 5.3 *Full preserves NProg.*

Proof: Observe that no transformation rule introduces new literals into the conclusion. □

Theorem 5.4 *Norm is complete and independent for NProg.*

Proof: To prove completeness it suffices to show that a *Norm*-reduced program which is not in solved form has no stable models. Then we will eventually obtain the solved forms for all stable models since every strategy has to apply some rule for non-reduced programs, and since *Norm* is terminating and sound.

Suppose P is *Norm*-reduced. Then Split assures that for every proposition A either $A \to$, $\to A$ or $\sim A \to$ is in P. Assume that for each proposition there exists one clause of form $A \to$ or $\to A$, but that P is not in solved form because P contains some additional clause C. If $C = \Box$ then $Stab(P) = \emptyset$ and we are done. P doesn't contain both $A \to$ and $\to A$ since Reduction-true is not applicable. Hence the premise of C cannot be empty, and we can apply some rule according to the following table, contradicting that P is reduced.

	$A \to$	$\to A$
$A, \Gamma \to \Delta$	Subsumption-false	Reduction-true
$\sim A, \Gamma \to \Delta$	GL-Reduction	GL-Subsumption

Hence we may now assume that the set of all propositions A such that P contains neither $A \rightarrow$ nor $\rightarrow A$ is nonempty. Let us call this set U. From the table above we also know that the premises of clauses in P which are not of form $A \rightarrow$ or $\rightarrow A$ contain only propositions from U, since otherwise some transformation is possible. Since the Split rule is not applicable, P contains a clause $\sim A \rightarrow$ for each A in U. Hence P has the form

$$\underbrace{\begin{array}{c} L_{11},\dots,L_{1k_1} \rightarrow \Delta_1 \\ \vdots \\ L_{n1},\dots,L_{nk_n} \rightarrow \Delta_n \end{array}}_{P'} \qquad \underbrace{\begin{array}{c} \sim B_1 \rightarrow \\ \vdots \\ \sim B_m \rightarrow \end{array}}_{P_U} \qquad \underbrace{\begin{array}{c} \rightarrow B'_1 \\ \vdots \\ \rightarrow B'_{m'} \end{array}}_{P_T} \qquad \underbrace{\begin{array}{c} B''_1 \rightarrow \\ \vdots \\ B''_{m''} \rightarrow \end{array}}_{P_F}$$

where $n, m', m'' \geq 0$, $m \geq 1$, $|\Delta_i| \leq 1$, $k_i + |\Delta_i| \geq 2$, $U = \{B_1, \dots, B_m\}$, $T = \{B'_1, \dots, B'_{m'}\}$ and $F = \{B''_1, \dots, B''_{m''}\}$ are pairwise disjoint, and all L_{ij} are of form B or $\sim B$ for B in U. Then P has at most the stable model $I = T \cup U$. $P_1 = GL_I(P)$ eliminates all clauses with negative literals. We will show that $J = T$ is a model of P_1. It satisfies $GL_I(P_T) = P_T$, $GL_I(P_F) = P_F$ and $GL_I(P_U) = \emptyset$. To see that J satisfies $GL_I(P')$, observe that any clause in $GL_I(P')$ has at least one positive premise from U. We conclude that I is not minimal and $\mathit{Stab}(P) = \emptyset$.

To prove independence we exhibit for each rule r in *Norm* a program in *NProg* which is $(\mathit{Norm} - \{r\})$-reduced, not in solved form and has a stable model.

Rule	Program	Stable models
Reduction-true (RedT)	$\{p \rightarrow q; \rightarrow p; \rightarrow q\}$	$\{p, q\}$
GL-Reduction (RedGL)	$\{\sim p \rightarrow q; p \rightarrow; \rightarrow q\}$	$\{q\}$
Subsumption-false (SubF)	$\{p, q \rightarrow; p \rightarrow; q \rightarrow\}$	$\emptyset$
GL-Subsumption (SubGL)	$\{\sim p \rightarrow; \rightarrow p\}$	$\{p\}$
Split	$\{\sim p \rightarrow q; \sim q \rightarrow p\}$	$\{p\}, \{q\}$

□

Theorem 5.5 *Disj is complete and independent for Prog.*

Proof: As in the completeness proof for *Norm* we will show that a *Disj*-reduced program which is not in solved form has no stable model. Suppose P is a *Disj*-reduced program. Then it is also *Norm*-reduced, and has the following form.

$$\underbrace{\begin{array}{c} L_{11},\dots,L_{1k_1} \rightarrow \Delta_1 \\ \vdots \\ L_{n1},\dots,L_{nk_n} \rightarrow \Delta_n \end{array}}_{P'} \qquad \underbrace{\begin{array}{c} \sim B_1 \rightarrow \\ \vdots \\ \sim B_m \rightarrow \end{array}}_{P_U} \qquad \underbrace{\begin{array}{c} \rightarrow B'_1 \\ \vdots \\ \rightarrow B'_{m'} \end{array}}_{P_T} \qquad \underbrace{\begin{array}{c} B''_1 \rightarrow \\ \vdots \\ B''_{m''} \rightarrow \end{array}}_{P_F}$$

where $n, m', m'' \geq 0$, $m \geq 1$, $k_i + |\Delta_i| \geq 2$, $U = \{B_1, \dots, B_m\}$, $T = \{B'_1, \dots, B'_{m'}\}$ and $F = \{B''_1, \dots, B''_{m''}\}$ are pairwise disjoint, and all L_{ij} are of form B or $\sim B$ for B in U. Moreover, RedF ensures that $\Delta_i \subseteq T \cup U$. Then P has at most the stable model $I = T \cup U$ and $GL_I(P) = GL_I(P') \cup P_T \cup P_F$, where $GL_I(P')$ contains exactly the negation-free clauses of P'.

Now suppose Disjunction is not applicable. If $k_i > 0$ for all i then $J = T$ is a model of $GL_I(P)$, hence I is not minimal.

If there exists a clause with $k_i = 0$, but $\tilde{P} \cup \{\rightarrow A_1, \dots, A_l\} \not\models \{\rightarrow A_1; \dots; \rightarrow A_l\}$ then there exists an interpretations M of $\tilde{P} \cup \{\rightarrow A_1, \dots, A_l\}$ such that $A_j \notin M$ for some $1 \leq j \leq l$. Because of P_U all propositions $\tilde{B}$ for B in U are false in M. Let $J = M \cap \mathit{Prop}$, then $J \models GL_I(P)$, $J \subseteq (T \cup U) \setminus \{A_j\} \subset T \cup U = I$ and I is not minimal. We conclude $\mathit{Stab}(P) = \emptyset$.

For independence we augment the proof for the non-disjunctive case by programs which are $(\mathit{Disj} - \{r\})$-reduced for $r \in \{\text{RedF}, \text{Disj}\}$. All the programs of the previous proof are reduced with respect to these rules, hence they carry over to this proof.

Rule	Program	Stable models
Reduction-false (RedF)	$\{\rightarrow p, q; p \rightarrow; \rightarrow q\}$	$\{q\}$
Disjunction (Dis)	$\{\rightarrow p, q; p \rightarrow q; q \rightarrow p; \sim p \rightarrow; \sim q \rightarrow\}$	$\{p, q\}$

□

6 Further work

Datalog. For the case of Datalog programs it would be desirable to avoid explicitly representing all false propositions and all instances of rules. This requires a more general notion of solved form and some modifications of the procedure since the original clauses will not be subsumed by their instances arising during computation. Moreover, Split and Disjunction may only be used on ground atoms, otherwise they are not sound. For the case of range-restricted programs, where all variables of a clause appear in its positive premises, this can be guaranteed.

Querying. Normally the interest is not so much in computing models but in query answering. It is possible to use a refutational approach for stable semantics. Consider a query $Q = A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m$ and let $\mathit{neg}(Q) = A_1, \ldots, A_n, \sim B_1, \ldots, \sim B_m \rightarrow$. Then for any program P

$$P \models_{\mathit{Stab}} Q \quad \text{iff} \quad \mathit{Stab}(P \cup \{\mathit{neg}(Q)\}) = \emptyset$$

It remains to develop a procedure which is specifically tailored to query answering. For that purpose it is not necessary to actually compute the stable models, but it suffices to test for satisfiability.

7 Conclusion

We have presented a new procedure for computing stable models of normal disjunctive logic programs. In analogy to the well-known Davis-Putnam procedure for classical propositional logic we used case analysis on the truth value of a proposition. It was crucial to allow clauses with empty heads. Then $\sim A \rightarrow$ could represent that A is constrained to be true, without justifying A. Together with $A \rightarrow$ this allowed case analysis. We have shown the soundness of several transformation rules on programs, which allow to transform a program into an equivalent one. These may be interesting in their own right, either to develop different procedures for reasoning with stable models, or because they provide more insight into the nature of stable semantics. For instance, Factoring allowed to notice that $\sim p \rightarrow p$ is equivalent to $\sim p \rightarrow$, hence its only effect is to constrain p to be true. From sets of sound rules we built our procedures and showed them to be complete, in the sense that they enumerate all stable models and terminate. We showed that for completeness only a subset of the rules was needed, while the others could be useful to improve performance. An actual implementation is free to choose which of these additional rules it uses.

We presented our procedure as a set of transformation rules, which resembled the standard method for presenting unification algorithms. Rules were sound in the sense that they preserved the set of stable models. The method allowed for concise presentation, flexible choice of a control strategy and simple correctness proofs.

Acknowledgements I would like to thank Harald Ganzinger for his helpful remarks on an earlier version of this report and Ullrich Hustadt and Yannis Dimopoulos for many discussions and for their general support.

References

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM* **7**: 201–215.

Eiter, T. and Gottlob, G. (1993). Complexity results for disjunctive logic programming and applications to nonmonotonic logics. *Proc. Int. Logic Programming Symp.*, pp. 266–278.

Eshghi, K. (1990). Computing stable models by using the ATMS. *Proc. AAAI-90*, Boston, pp. 272–277.

Fernández, J. A. and Minker, J. (1992). Disjunctive deductive databases. *Int. Conf. on Logic Programming and Automated Reasoning*, Springer LNCS 624, St. Petersburg, pp. 332–356.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. *Proc. Logic Programming Conf.*, Seattle.

Inoue, K., Koshimura, M. and Hasegawa, R. (1992). Embedding negation as failure into a model generation theorem prover. In D. Kapur (ed.), *11th International Conference on Automated Deduction*, Springer LNAI 607, Saratoga Springs, NY, pp. 400–415.

Marek, W. and Subrahmanian, V. S. (1992). The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theoretical Computer Science* **103**: 365–386.

Saccà, D. and Zaniolo, C. (1990). Stable models and non-determinism in logic programs with negation. *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, TN, pp. 205–229.

A Consequence relations

For the sake of the soundness proofs we introduce a consequence relation $\mid\!\sim$ between programs. For other purposes $\mid\!\sim$ is of little use, since it is not closed with respect to (cautious) cut. Define $\mid\!\sim$ by $P_1 \mid\!\sim P_2$ if and only if for all interpretations I and J such that $I \models GL_I(P_1)$, $J \subseteq I$, and $J \in Min(GL_I(P_1))$ we have $J \models GL_I(P_2)$.

Lemma A.1 *Let P_1 and P_2 be programs. If $P_1 \mid\!\sim P_2$ and $P_2 \mid\!\sim P_1$ then $Stab(P_1) = Stab(P_2)$.*

Proof: Let P_1 and P_2 be programs and let I be an interpretation. We have to show that I is a minimal model of $P_1' = GL_I(P_1)$ if and only if I is a minimal model of $P_2' = GL_I(P_2)$.

For the only-if direction suppose I is a minimal model of P_1'. Then by $P_1 \mid\!\sim P_2$ with $J = I$ we get $I \models P_2'$. Suppose I is not a minimal model of P_2', then there exists a minimal model J of P_2' with $J \subset I$. By $P_2 \mid\!\sim P_1$ we get $J \models P_1'$, in contradiction to our assumption that I is a minimal model of P_1'. We conclude that I is a minimal model of P_2'. The argument for the other direction is exactly symmetric. □

Let $CTaut = \{\, A, \tilde{A} \rightarrow \mid A \in Prop \,\}$.

Lemma A.2 *Let P_1 and P_2 be programs. Then $\tilde{P}_1 \cup CTaut \models \tilde{P}_2$ implies $P_1 \mid\!\sim P_2$.*

Proof: Assume that I and J are interpretations such that $J \subseteq I$ and $J \models GL_I(P_1)$, and combine them into an interpretation $M = J \cup \{\, \tilde{A} \mid A \notin I \,\}$ suitable for $\tilde{P}_1$. We first show that $M \models \tilde{P}_1 \cup CTaut$. Suppose $C = A, \tilde{A} \rightarrow \in CTaut$. By $J \subseteq I$ $A \in J$ implies $A \in I$, hence $A \in M$ implies $\tilde{A} \notin M$ and C is true in M. Now suppose that C is a clause in P_1. If GL_I eliminates C then $A \in I$ for some literal $\sim A$ in C. Thus $\tilde{A} \notin M$ and $\tilde{C}$ is true in M. If on the other hand $GL_I(C) = \{C'\}$, we have $J \models C'$. Since C' is negation-free this implies $M \models C'$, and since C' is a subclause of C we get $M \models \tilde{C}$. Thus by assumption $M \models \tilde{P}_2$.

Now let C be any clause in P_2. If $GL_I C = \emptyset$ we are done, so assume $GL_I C = \{C'\}$. In this case $A \notin I$ for all $\sim A$ in C. Then $\tilde{A} \in M$ for all $\tilde{A}$ in $\tilde{C}$, which implies $M \models \tilde{C}'$. Since C' is negation-free $J \models C'$ follows and we are done. □

Lemma A.3 *Let P_1, P_2 and P_3 be programs.*

1. *$P_1 \mid\!\sim P_1$.*
2. *$P_1 \mid\!\sim P_2$ and $P_1 \mid\!\sim P_3$ imply $P_1 \mid\!\sim P_2 \cup P_3$.*
3. *$P_1 \mid\!\sim P_2$ and $P_3 \subseteq P_2$ imply $P_1 \mid\!\sim P_3$.*
4. *$P_2 \subseteq P_1$ implies $P_1 \mid\!\sim P_2$.*
5. *$P_1 \mid\!\sim P_2$ if and only if $P_1 \mid\!\sim P_1 \cup P_2$.*

Proof: (1), (2) and (3) are trivial from the definition of $\mid\!\sim$. (4) is a consequence of lemma A.2. The if part of (5) follows from (1) and (3), while the only-if part follows from (1) and (2). □

B Soundness proofs

Lemma B.1 *Let P_1 and P_2 be programs. If $P_1 \vdash P_2$ by Tautology elimination then $P_1 \mid\!\sim P_2$ and $P_2 \mid\!\sim P_1$.*

Proof: This is an immediate consequence of lemma A.2. □

Lemma B.2 *Let P_1 and P_2 be programs. If $P_1 \vdash P_2$ by Resolution then $P_1 \mid\!\sim P_2$ and $P_2 \mid\!\sim P_1$.*

Proof: Let P_1 and P_2 be programs such that $P_1 \vdash P_2$ by Resolution. Then $P_1 = \{C_1, C_2\} \uplus P$ and $P_2 = \{C_1, C_2, C_3\} \uplus P$ for some program P and clauses C_1, C_2 and C_3 where C_3 is a resolvent of C_1 and C_2. Since $P_1 \subseteq P_2$ we have $P_2 \mid\!\sim P_1$. For $P_1 \mid\!\sim P_2$ it suffices by lemma A.3 to show $P_1 \mid\!\sim C_3$. For Standard resolution this is a consequence of lemma A.2.

For Constraint resolution we show that for all interpretations I and J with $I \models GL_I(P_1)$ and $J \subseteq I$ such that $J \models GL_I(P_1)$ we have $J \models GL_I(\{C_3\})$. If GL_I eliminates C_3 we are done. Suppose this is not the case.

For Constraint resolution $C_1 = \sim A, \Gamma \to \Delta$, $C_2 = A, \Lambda \to \Pi$ and $C_3 = \Gamma, \Lambda, \sim\cdot\Pi \to \Delta$. Since GL_I doesn't eliminate C_3 we have $\Pi \cap I = \emptyset$. Then GL_I eliminates neither $\Gamma \to \Delta$ nor $\Lambda \to \Pi$. Let $GL_I(\Gamma \to \Delta) = \Gamma' \to \Delta$ and $GL_I(\Lambda \to \Pi) = \Lambda' \to \Pi$. Then $GL_I(C_3) = \Gamma', \Lambda' \to \Delta$. If $A \in I$ then, because I is a model of $GL_I(\{C_1, C_2\})$ and $\Pi \cap I = \emptyset$, we have $\Lambda' \not\subseteq I$. Via $J \subseteq I$ we get $\Lambda' \not\subseteq J$. We conclude $J \models GL_I(C_3)$. If on the other hand $A \notin I$ then $A \notin J$ and $J \models GL_I(\Gamma \to \Delta)$, which implies $J \models GL_I(C_3)$. □

Lemma B.3 *Let P_1 and P_2 be programs. If $P_1 \vdash P_2$ by subsumption then $P_1 \mid\!\sim P_2$ and $P_2 \mid\!\sim P_1$.*

Proof: Let P_1 and P_2 be programs such that $P_1 \vdash P_2$ by subsumption. Then $P_1 = \{C_1, C_2\} \uplus P$ and $P_2 = \{C_1\} \uplus P$ for some program P and clauses C_1 and C_2 where C_1 subsumes C_2. Since $P_2 \subseteq P_1$ we trivially have $P_1 \mid\!\sim P_2$. For $P_2 \mid\!\sim P_1$ it suffices by lemma A.3 to show $P_2 \mid\!\sim C_2$, and in turn by lemma A.2 to show $\tilde{C_1} \cup \mathit{CTaut} \models \tilde{C_2}$. The clauses have the form $\tilde{C_1} = \Gamma, \tilde{A_1}, \ldots, \tilde{A_n} \to B_1, \ldots, B_m, \Delta$ and $\tilde{C_2} = \Gamma', \tilde{A_1}, \ldots, \tilde{A_n}, \tilde{B_1}, \ldots, \tilde{B_m} \to \Delta'$ with $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$. By a series of classical resolutions of $\tilde{C_1}$ with $\{B_1, \tilde{B_1} \to; \ldots; B_m, \tilde{B_m} \to\}$ we get $\Gamma, \tilde{A_1}, \ldots, \tilde{A_n}, \tilde{B_1}, \ldots, \tilde{B_m} \to \Delta$ which implies $\tilde{C_2}$. □

Lemma B.4 *Let P_1 and P_2 be programs. If $P_1 \vdash P_2$ by Factoring then $P_1 \mid\!\sim P_2$ and $P_2 \mid\!\sim P_1$.*

Proof: Let P_1 and P_2 be programs such that $P_1 \vdash P_2$ by Factoring. Then $P_1 = \{C_1\} \uplus P$ and $P_2 = \{C_1, C_2\} \uplus P$ for some program P and clauses $C_1 = \sim A, \Gamma \to A, \Delta$ and $C_2 = \sim A, \Gamma \to \Delta$. Since $P_1 \subseteq P_2$ we trivially have $P_2 \mid\!\sim P_1$. For $P_1 \mid\!\sim P_2$ it suffices by lemma A.3 to show $P_1 \mid\!\sim C_2$, which by lemma A.2 is a consequence of $\tilde{C_1} \cup \mathit{CTaut} \models \tilde{C_2}$. This is can be demonstrated by classical resolution and factoring. □

Lemma B.5 *Let P_1 and P_2 be programs. If $P_1 \vdash P_2$ by Default negation then $P_1 \mid\!\sim P_2$ and $P_2 \mid\!\sim P_1$.*

Proof: Let P_1 and P_2 be programs such that $P_1 \vdash P_2$ by Default negation. Then $P_2 = \{C\} \uplus P_1$ for a clause $C = A \to$. Since $P_1 \subseteq P_2$ we have $P_2 \mid\!\sim P_1$. For $P_1 \mid\!\sim P_2$ it suffices by lemma A.3 to show $P_1 \mid\!\sim C$. We show that for all interpretations I and J where $J \in \mathit{Min}(GL_I(P_1))$ we have $J \models GL_I(\{C\})$.

Suppose $A \in J$ and let $J' = J - \{A\}$. Clauses in P_1 which don't contain A are satisfied by J', since J is a model of P_1. Any clause in P_2 which contains A has the form $C = \Gamma, A \to \Delta$, since GL_I deletes all literals $\sim A$ and by assumption A does not appear in the conclusion. But since $A \notin J'$ we have $J' \models C$. Hence J is not minimal, in contradiction to the assumption. We conclude $A \notin J$ and $J \models P_1$. □

Lemma B.6 *Let P be a program and A a proposition. Then*

$$\mathit{Stab}(P \cup \{A \to\}) = \{\, I \mid I \in \mathit{Stab}(P) \text{ and } A \notin I \,\} \qquad \text{and}$$
$$\mathit{Stab}(P \cup \{\sim A \to\}) = \{\, I \mid I \in \mathit{Stab}(P) \text{ and } A \in I \,\}.$$

Proof: Let P be a program, I an interpretation and A a proposition. Consider the case where $A \in I$. We have to show that I is a minimal model of $GL_I(P)$ iff it is a minimal model of $GL_I(P \cup \{\sim A \rightarrow\})$. But from the definition of GL_I we get $GL_I(P \cup \{\sim A \rightarrow\}) = GL_I(P)$.

Now suppose $A \notin I$. It is easy to see that $I \in \mathit{Min}(GL_I(P))$ iff $I \in \mathit{Min}(GL_I(P) \cup \{A \rightarrow\})$ from the fact that I satisfies $A \rightarrow$. □

Lemma B.7 *Let P_1 and P_2 be programs. If $P_1 \vdash P_2$ by Disjunction then $P_1 \mid\!\sim P_2$ and $P_2 \mid\!\sim P_1$.*

Proof: The proof is immediate from the condition on the rule and lemmas A.3 and A.2.

Theorem 4.1 is a consequence of these lemmas and lemma A.1.