Towards Automating Duality

Chris Brink & Dov Gabbay & Hans Jürgen Ohlbach

MPI–I–93–220 May 1993

Author's Address

Chris Brink
Dept. of Mathematics
University of Cape Town
7700 Rondebosch
South Africa
email: cbrink@maths.uct.ac.za

Dov M. Gabbay
Imperial College of Science, Technology and Medicin
Dept. of Computing
Huxley Building, 180 Queen's Gate
London SW7 2AZ,
England
email: dg@doc.ic.ac.uk

Hans Jürgen Ohlbach
Max–Planck–Institut für Informatik
Im Stadtwald
D-6600 Saarbrücken 11
F. R. Germany
email: ohlbach@mpi-sb.mpg.de

Publication Notes

This paper has also appeared as a Research Report of the Department of Mathematics at the University of Cape Town.

Acknowledgements

This work was supported by the ESPRIT project 3125 MEDLAR, by the "Sonderforschungsbereich" 314, "Künstliche Intelligenz und wissensbasierte Systeme" of the German Research Council (DFG) and by the BMFT funded project LOGO (ITS 9102).

Parts of the paper were written while the third author was visiting the University of Cape Town.

The second author is a SERC Senior Research Fellow.

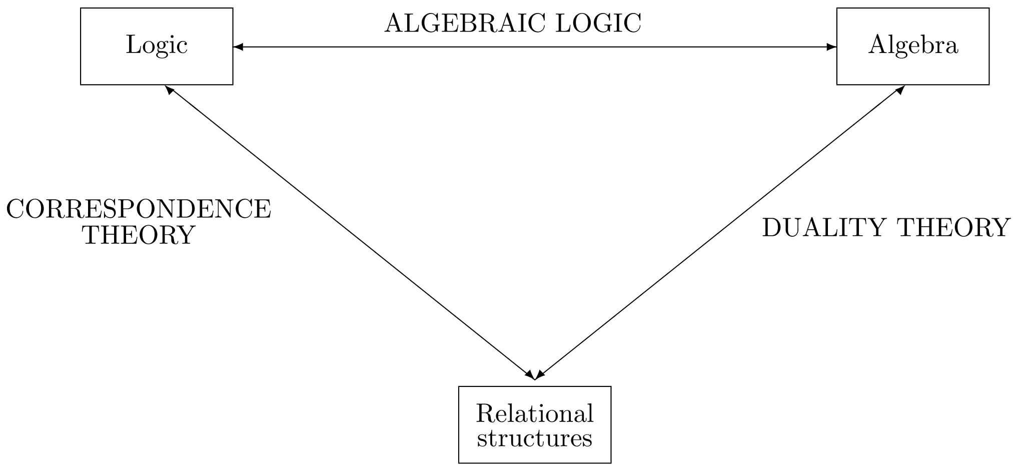
Abstract

Dualities between different theories occur frequently in mathematics and logic — between syntax and semantics of a logic, between structures and power structures, between relations and relational algebras, to name just a few. In this paper we show for the case of structures and power structures how corresponding properties of the two related structures can be computed fully automatically by means of quantifier elimination algorithms and predicate logic theorem provers. We illustrate the method with a large number of examples and we give enough technical hints to enable the reader who has access to the OTTER theorem prover to experiment herself.

Contents

1 Introduction

To any logic satisfying certain minimal requirements there corresponds both an algebra and a relational semantics, and the general picture of such relationships can be presented as in the figure below.



As a paradigm case, consider the modal logic S4. Its algebraic counterpart is the variety of closure algebras, related to the logic through the Lindenbaum/Tarski construction and an algebraic completeness result. The Kripke semantics of S4, on the other hand, is given by the theory of quasi-orders (reflexive transitive relations). Finally, quasi-orders and closure algebras are related to each other through a power construction first used in a (now famous but long neglected) paper of Jónsson and Tarski [JT51][JT52].

Algebraic Logic, broadly speaking, stands in the tradition of the algebraization of classical propositional logic as the variety of Boolean algebras. This has been generalised to many extensions of classical propositional logic (notably modal logics), yielding various *Boolean Algebras with Operators* ([Jón92]), and also to variations on classical propositional logics (e.g. intuitionistic logic, relevance logic, many-valued logics), typically yielding distributive lattices with operators. These propositional cases are fairly well understood (Blok and Pigozzi [BP89]), but the problem of algebraization of first- and higher-order logics has proved much more difficult. The *cylindric algebras* of Henkin, Monk and Tarski [HMT71] [HMT85], the *monadic algebras* of Halmos [Hal62] and the work of Craig [Cra74] are all attempts to present algebraic versions of first-order logic. For a comprehensive overview see Nemeti [Nem92].

Duality Theory, from the perspective of non-classical logic (Bull and Segerberg [BS84]), studies the relationship between the semantics of a logic and its algebra, and how to obtain each from the other. Typically, the semantics is defined with reference to some relational structure (sometimes called a *frame*, or *model structure*); the corresponding algebra is obtained by a power construction, and from the algebra the model structure can be recovered by imposing a relational structure on the set of ultrafilters. More generally, duality theory studies the relationship between algebras and relational structures without necessarily referring to any logic. The variety of *relation algebras* is a case in point. These algebras arose in response to the problem posed by Tarski [Tar41] of finding equational axioms that would capture the calculus of binary relations, in the same way as the axioms for Boolean algebras capture the calculus of sets. Already in [JT51] [JT52] it was established that relation algebras stand in a duality relationship to structures called *generalized Brandt groupoids*, whereas the relationship of algebras of relations to first-order logic was only fully presented by Tarski and Givant in [TG87]. A different but related perspective on duality is that of the topologist. Stone [Sto37] related Boolean algebras to topological spaces, and the study of certain lattices and their topological duals became a topic of study in its own right (Johnstone [Joh82]). Through the work of Priestley [Pri70] and Hansoul [Han83] this extends also to lattices *with operators*, which are dual to topological spaces endowed with relational structure. By this route the semantics of some logics are in full topological duality to their algebras.

Correspondence Theory (or *definability theory*) considers the classical definability of nonclassical formulae (specifically propositional *modal* formulae) when viewed as relational principles (Van Benthem [vB84]). The notion of expressing the 'meanings' of modalities in terms of a *possible-world semantics* goes back to Kripke [Kri59], who coded principles of logic as properties of an accessibility relation between possible worlds. More generally, the question arises which modal formulae define *first-order* relational conditions - and how do they do it? Conversely, which first-order relational conditions are modally definable?

The relevance of these well-established studies to Computer Science is gradually becoming clear. One obvious application is to so-called program logics (e.g. dynamic logic, Kozen [Koz81]), many of which are variations on modal or multi-modal logic. Such logics have been explicitly linked to Boolean algebras with operators (Pratt [Pra90]), and suggestions have been made on the use of relation algebras for program specification (Jónsson [Jón], Hoare and He Jifeng [HJ87]). *Denotational semantics* of programming languages seems another promising area, particularly in view of the presentation of *domain theory* in logical form (Abramsky [Abr87]), and the constructions required for *powerdomains*. Finally, formalisms such as *bilattices* (Ginsberg [Gin88]) and various deontic and epistemic logics proposed for AI research again fit the triangular pattern of logic-algebra-semantics. One recent example is that the independently-conceived system KL-ONE for knowledge representation (Brachman and Schmolze [BS85]) can be viewed as having a semantics of relations interacting with sets, and thus to have both a modal logic presentation (Schild [Sch91]) and an algebraic one (Brink and Schmidt [BS92]).

The fact that there are well-established areas of research relating logic, algebra and semantics shows that translation of individual formulae from a logical to an algebraic (equational) and to a structural (first-order) version is not a trivial matter. Many such translations are known; all of these have been found by traditional pencil-and-paper methods. Recently, however, Gabbay and Ohlbach [GO92] have proposed to inject a measure of *automated reasoning* into the area, by the use of an algorithm (called SCAN) for quantifier elimination in second-order logic. They give a number of examples from *correspondence theory*, showing how Hilbert axioms may by means of SCAN be translated into first-order properties of the accessibility relation.

The aim of this paper is to continue research in this area by considering the automation of *duality theory*, in both directions. More particularly, it is to marry the use of SCAN to the concept of *power structures*, presented in Brink [Bri92] as a useful cross-disciplinary unifying concept. Being 'more mathematical', we hope in this way to make the proposal of automation accessible also to mathematicians unfamiliar with the intricacies of non-classical logics. It turns out that the methods for finding correspondences between Hilbert axioms and properties of the relational structures (correspondence theory) and for finding correspopnding properties for structures and power structures (duality theory) are almost identical. Therefore this paper is as relevant to correspondence theory as it is to duality theory.

The mathematical theory of power structures is introduced in the next section. The 'duality algorithms' make use of recent developments in resolution based theorem proving. Since the paper also addresses mathematicians and logicians who might not be familiar with automated theorem proving, we give a very brief overview on these topics in section 3, and in section 6 we introduce the theorem prover OTTER. OTTER serves as our vehicle for making the algorithms work on a computer. The general framework for automating duality is presented in section 4. The kernel of the algorithms, the quantifier elimination algorithm, is introduced in section 5. Eventually we present a large number of examples.

2 Power Structures and Dualities

The theory of Boolean algebras with operators was introduced in Jónsson and Tarski [JT51],[JT52]. The operators on the elements of the Boolean algebra are assumed to be additive in each argument. One way in which a Boolean algebra with operators arises is as the *power algebra* of a relational structure. If a relational structure is defined over some set A, then its power algebra is defined over the power set of A. This power algebra is a Boolean algebra of sets, with the usual set-theoretic operations $\cup$, $\cap$ and $'$, but with additional operators on the Boolean algebra which are the *power operations* of the relations defined over A.

Definition 2.1 *For any set A, and any (n+1)-ary relation $R \subseteq A^{n+1}$, the power operation $R^{\uparrow} : \mathcal{P}(A)^n \to \mathcal{P}(A)$ is defined by:*

$$R^{\uparrow}(X_0, \ldots, X_{n-1}) = \{x_n \mid (\exists x_0 \in X_0) \ldots (\exists x_{n-1} \in X_{n-1})[Rx_0 \ldots x_n]\},$$

for every $X_0, \ldots, X_{n-1} \subseteq A$.

For any relational structure $\mathcal{A} = (A; R_1, \ldots, R_m)$, the power algebra $\mathcal{P}(\mathcal{A})$ is defined by:

$$\mathcal{P}(\mathcal{A}) = (\mathcal{P}(A); R_1^{\uparrow}, \ldots, R_m^{\uparrow}).$$

For example, if R is a binary relation, then $R^{\uparrow} : \mathcal{P}(A)^n \to \mathcal{P}(A)$ is defined by $R^{\uparrow}(X) = \{y \mid (\exists x \in X)[Rxy]\}$. Adding the operations $\cup$, $\cap$ and $'$ to the power algebra $\mathcal{P}(\mathcal{A})$ yields a Boolean algebra with operators $R_1^{\uparrow}, \ldots, R_m^{\uparrow}$. Jónsson and Tarski further proved that all Boolean algebras with operators arise in this fashion, thus obtaining a representation theorem.

Theorem 2.2 *Any Boolean algebra with operators is isomorphic to a subalgebra of the power algebra of some relational structure.*

An operator on sets is called *normal* if whenever one of the arguments of the operator is the empty set, then so is the outcome. A Boolean algebra with operators is normal if all its operators are normal. For these algebras, the inverse of the power construction of definition 2.1 provides a mechanism to obtain the underlying relational structure of the Boolean algebra with operators.

Definition 2.3 *For any set A, and any n-ary operator $F : \mathcal{P}(A)^n \to \mathcal{P}(A)$, the underlying (n+1)-ary base relation $F^{\downarrow} \subseteq A^{n+1}$ is defined by:*

$$F^{\downarrow}x_0 \ldots x_{n-1}x_n \text{ iff } x_n \in F(\{x_0\}, \ldots, \{x_{n-1}\}).$$

For any Boolean algebra $\mathcal{P}(\mathcal{A})$ with operators $F_1, \ldots, F_m$, the underlying relational structure $\mathcal{A}$ is defined by:

$$\mathcal{A} = (A; F_1^{\downarrow}, \ldots, F_m^{\downarrow}).$$

Theorem 2.4 *For any relation $R \subseteq A^{n+1}$, $(R^{\uparrow})^{\downarrow} = R$, and for any normal and additive operator F:$\mathcal{P}(A)^n \to \mathcal{P}(A)$, $(F^{\downarrow})^{\uparrow} = F$.*

In the context of duality theory, the Jónsson/Tarski construction provides a mechanism to translate between second-order properties defining the possible world semantics of a logic, and properties of operators defined on the Lindenbaum/Tarski algebra of the logic. The next two lemmas list a number of such translations, the first between properties of a unary operation and its corresponding binary relation, and the second between properties of a binary operation and its corresponding ternary relation. Traditional proofs can be found in [Bri89]; our machine-generated proofs appear in section 8.

Lemma 2.5 *Let $F : \mathcal{P}(U) \to \mathcal{P}(U)$ be normal and completely additive, and let $R \subseteq U^2$ be $F^{\downarrow}$, then properties of R correspond to properties of F as (ii) to (i) below. Conversely, let $R \subseteq U^2$ be any relation, and let $F = R^{\uparrow}$, then $F : \mathcal{P}(U) \to \mathcal{P}(U)$ is normal and completely additive, and properties of F correspond to properties of R as (i) to (ii) below.*

(a) (i) $\forall X \subseteq U : \; X \neq \emptyset \Rightarrow F(X) \neq \emptyset$,
    (ii) Domain $(R) = U$;
(b) (i) $\forall X \subseteq U : \; X \subseteq F(X)$,
    (ii) R is reflexive over U;
(c) (i) $\forall X \subseteq U : \; F(X) \subseteq X$,
    (ii) R is the identity relation over a subset of U;
(d) (i) $F : \mathcal{P}(U) \to \mathcal{P}(U)$ is the identity function,
    (ii) $R \subseteq U^2$ is the identity relation;
(e) (i) $E \subseteq U$ is a fixed point of $F : F(E) = E$,
    (ii) $(\exists e \in E)[Rex]$ iff $x \in E$;
(f) (i) $\forall X \subseteq U : \; F(F(X)) \subseteq F(X)$,
    (ii) R is transitive;
(g) (i) $\forall X \subseteq U : \; F(X) \subseteq F(F(X))$,
    (ii) R is dense $[\forall x, y \in U : Rxy \Rightarrow (\exists z)[Rxz \& Rzy]]$;
(h) (i) $\forall X, Y \subseteq U : \; F(X) \bigcap Y = \emptyset$ iff $X \bigcap F(Y) = \emptyset$,
    (ii) R is symmetric;
(i) (i) F maps singletons onto singletons,
    (ii) R is a unary operation over U;
(j) (i) F is an involution over $\mathcal{P}(U)$,
    (ii) R is an involution over U.

Notice that the function F is the algebraic counterpart of the $\Diamond$–operator in modal logic. This becomes clear when we compare the definition of F with the semantics of $\Diamond$.

$$x \in F(X) \Leftrightarrow \exists x_0 \; R(x_0, x) \wedge x_0 \in X$$
$$x \models \Diamond X \Leftrightarrow \exists x_0 \; R(x, x_0) \wedge x_0 \models X$$

The only difference is that the arguments of the R–relation are exchanged, which is due to different tradition in the different communities.

Lemma 2.6 *Let $F : \mathcal{P}(U)^2 \to \mathcal{P}(U)$ be normal and completely additive, and let $R \subseteq U^3$ be $F^{\downarrow}$, then properties of R correspond to properties of F as (ii) to (i) below. Conversely, let $R \subseteq U^3$ be any relation, and $F = R^{\uparrow}$, then $F : \mathcal{P}(U)^2 \to \mathcal{P}(U)$ is normal and completely additive, and properties of F correspond to properties of R as (i) to (ii) below.*

(a) (i) F has no divisors of zero $[\forall X, Y \subseteq U : \; F(X, Y) = \emptyset \Rightarrow X = \emptyset$ or $Y = \emptyset]$,
    (ii) $\forall x, y \; \in U, \exists z \in U$ such that $Rxyz$;
(b) (i) F is commutative $[\forall X, Y \subseteq U : \; F(X, Y) = F(Y, X)]$,
    (ii) F is $(1,2)$-symmetric $[\forall x, y, z \; \subseteq U : \; Rxyz \Rightarrow Ryxz]$;
(c) (i) F is upper semi-idempotent $[\forall X \subseteq U : \; X \subseteq F(X, X)]$,
    (ii) R is totally reflexive $[\forall x \subseteq U : \; Rxxx]$;
(d) (i) F is lower semi-idempotent $[\forall X \subseteq U : \; F(X, X) \subseteq X]$,
    (ii) R is 3-prime $[\forall x, y, z \; \in U : \; Rxyz \Rightarrow z = x$ or $z = y]$;
(e) (i) F is idempotent $[\forall X \subseteq U : \; F(X, X) = X]$,
    (ii) R is totally reflexive and 3-prime;
(f) (i) F associates from left to right $[\forall X, Y, Z \; \subseteq U : \; F(F(X, Y), Z) \subseteq F(X, F(Y, Z))]$,
    (ii) R^2 associates from left to right
    $[\forall x, y, z, u \; \in U : \; (\exists v)[Rxyv \wedge Rvzu] \Rightarrow (\exists w)[Rxwu \wedge Ryzw]]$
    Abbreviation: $[R^2(xy)zu \Rightarrow R^2x(yz)u]$;
(g) (i) F associates from right to left $[\forall X, Y, Z \; \subseteq U : \; F(X, F(Y, Z)) \subseteq F(F(X, Y), Z)]$,
    (ii) R^2 associates from right to left
    $[\forall x, y, z, u, \; \in U : \; (\exists w)[Rxwu \wedge Ryzw] \Rightarrow (\exists v)[Rxyv \wedge Rvzu]]$
    Abbreviation: $[R^2x(yz)u \Rightarrow R^2(xy)zu]$;
(h) (i) F is associative,

(ii) R^2 is associative $[R^2(xy)zu$ iff $R^2x(yz)u]$;

(i) (i) $E \subseteq U$ is a left identity of F $[\forall X \subseteq U: \ F(E,X) = X]$,

(ii) $E \subseteq U$ is a set of left identities of R $[\forall x, y \subseteq U: \ (\exists e \in E)[Rexy]$ iff $x = y]$.

Consider for example the modal logic $S4$. Its Kripke semantics is given by the theory of quasi-orders, sometimes called $S4$-model structures, while its algebraic counterpart is given by the variety of closure algebras. From any $S4$-model structure $\mathcal{A} = (A, R)$, with set of worlds A and quasi-order R, one obtains a corresponding closure algebra $\mathcal{P}(\mathcal{A}) = (\mathcal{P}(A), R^{\uparrow})$ with closure operator $R^{\uparrow}$. Properties of R then translate to properties of $R^{\uparrow}$ as in lemma 2.5. For any closure algebra $\mathcal{P}(\mathcal{A}) = (\mathcal{P}(A), F)$ over the power set of A, one obtains a corresponding $S4$-model structure $\mathcal{A} = (A, F^{\downarrow})$ with quasi-order $F^{\downarrow}$. Properties of F then translate to properties of $F^{\downarrow}$ as in lemma 2.5.

As a second example, consider the relevance logic $\mathcal{R}^{\neg}$, the logic obtained from the (standard) relevance logic $\mathcal{R}$ ([AB75]) by adding a Boolean negation operation $\neg$. Its Kripke semantics is given by the theory of $\mathcal{R}^{\neg}$-relational structures, while its algebraic counterpart is given by the class of $\mathcal{R}^{\neg}$-algebras. The following definitions are from [Bri89].

Definition 2.7 *An $\mathcal{R}^{\neg}$-model structure is a relational structure $\mathcal{U} = (U; R, ^*, E)$, where $R \subseteq U^3$, $^* : U \to U$ and $E \subseteq U$ are such that:*

(a) R is totally reflexive: $Raaa$,

(b) (1,2)-symmetric: $Rabc \Rightarrow Rbac$,

(c) and has identity elements in E: $(\exists e \in E)[Reab]$ iff $a = b$,

(d) R^2 is associative: $(\exists x)[Rabx \,\&Rxcd]$ iff $(\exists y)[Rayd \,\&Rbcy]$,

(e) * is an involution: $a^{**} = a$,

(f) and R and * obey the rule: $Rabc \Rightarrow Rac^*b^*$.

Definition 2.8 *An $\mathcal{R}^{\neg}$-algebra is an algebra $\mathcal{A} = (A; \vee, \neg, 1, \circ, \sim, e)$ such that:*

(a) $(A; \vee, \neg, 1)$ is a Boolean algebra,

(b) $\sim$ is an involution: $\sim\sim a = a$,

(c) with the De Morgan properties: $\sim (a \vee b) = \sim a \wedge \sim b$,

(d) $(A; \circ, e)$ is a commutative monoid,

(e) which is lattice-ordered: $a \circ (b \vee c) = (a \circ b) \vee (a \circ c)$,

(f) $\circ$ is upper semi-idempotent: $a \leq a \circ a$,

(g) and has the antilogism property: $a \circ b \leq c$ iff $(a \circ \sim c) \leq \sim b$.

In order to show that $\mathcal{R}^{\neg}$-algebras arise from $\mathcal{R}^{\neg}$-model structures by way of the power construction, the operations on $\mathcal{R}^{\neg}$ must be additive. Since the De Morgan negation $\sim$ is not additive, definition 2.8 is reformulated using the star operation $^\star : A \to A$, defined by

$$a^\star = \sim \neg a,$$

instead of $\sim$. The new definition agrees with definition 2.8 above in all respects except these: $\sim$ is replaced by *, and axioms (b), (c) and (g) become, respectively:

(b)$'$ * is an involution: $a^{\star\star} = a$,

(c)$'$ which is additive: $(a \vee b)^\star = a^\star \vee b^\star$,

(g)$'$ $\circ$ has the property: $a \circ (b^\star) \leq \neg c$ iff $a \circ (c^\star) \leq \neg b$.

The advantage of the alternative definition is that it turns an $\mathcal{R}^{\neg}$-algebra into a Boolean algebra with operators. Lemma 2.6 then provides the translation between properties of R and properties of $\circ$, and between properties of * and properties of *.

3 A Brief Tutorial about Logic and Theorem Proving

The methods we are going to present in this paper rely heavily on recent developments in predicate logic and resolution based theorem proving. Since the paper addresses mathematicians and logicians who might not be that familiar with resolution based theorem proving, we give a very brief overview on predicate logic and resolution based theorem proving. For more information we refer to the textbooks in this area [CL73, Lov78, BB91, WOLB91].

3.1 Predicate Logic

Predicate Logic, as any other logic, is defined via specifying the syntax of the formulae and characterizing the true formulae (tautologies) in some way. This can be done either by giving a model theoretic semantics or by means of a procedure that characterizes the true formulae syntactically. An axiomatization in the Hilbert style, a Gentzen calculus, a Resolution calculus are some of the syntax oriented options. If both a semantics and a syntactic characterization of true formulae are given, they should agree on the same formulae.

First–order predicate logic (PL1) is an extension of propositional logic. The logical connectives are $\neg$ (negation), $\wedge$ (and), $\vee$ (or), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence). In addition we have the universal quantifier $\forall$ and the existential quantifier $\exists$. The non–logical syntax elements are *variables* (usually written x, y, z, u, v, w), *constant symbols* (written a, b, c, d), function symbols (written f, g, h, k) and predicate symbols (written P, Q, R). Function and predicate symbols have a fixed *arity.* From variable, constant and function symbols, *terms* are built according to the rules: (i) variable and constant symbols are terms (ii) if $t_1, \ldots, t_n$ are terms and f is a function symbol of arity n then $f(t_1, \ldots, t_n)$ is a term. If $t_1, \ldots, t_n$ are terms and P is a predicate symbol of arity n then $P(t_1, \ldots, t_n)$ is an *atom.* A *literal* is either an atom $P(t_1, \ldots, t_n)$ or a negated atom $\neg P(t_1, \ldots, t_n)$.

PL1 formulae are built according to the rules (i) each atom is a formula, (ii) if Φ and Ψ are formulae, x is a variable then $\neg\Phi$, $\Phi \wedge \Psi$, $\Phi \vee \Psi$, $\Phi \Rightarrow \Psi$, $\Phi \Leftrightarrow \Psi$, $\forall x\ \Phi$ and $\exists x\ \Phi$ are formulae.

While first–order predicate logic does not allow quantification over functions and predicates, this is allowed in second–order predicate logic. In second–order predicate logic a formula like $\forall P\ \exists f\ \forall x\ P(f(x), P, f)$ is allowed.

The semantics of first–order predicate logic, developed by Alfred Tarski, assigns elements of a certain set ('domain', 'universe' of the interpretation) to variable and constant symbols, n–place functions to n–ary function symbols and n–place relations to n–ary predicate symbols. Terms are interpreted as function applications, atoms as functions to boolean values and logical connectives as boolean functions. The quantifiers are interpreted as expected. Each such assignment of mathematical objects to the syntactic symbols yields an *interpretation.* Each formula is either true or false in each particular interpretation. If it is true in an interpretation, this interpretation is called a *model* of the formula. *Tautologies* are formulae which are true in *all* interpretations. *Contradictions* are formulae which are *false* in all interpretations. A formula Φ *entails* Ψ (written $\Phi \models \Psi$) iff Ψ is true in all models of Φ. The semantics of second–order predicate logic is an extension of the semantics for PL1 in that variables can also be mapped to functions and relations (this is extremely brief, things are more complicated.)

There are various *normal forms* of PL1–formulae. The one we shall use in the sequel is the *conjunctive normal form* or *clause form.* A *clause* is simply a disjunction of literals $L_1 \vee \ldots \vee L_m$ (usually written as a set $L_1, \ldots, L_n$). An arbitrary PL1–formula can be transformed into a list (conjunction) of clauses by the following rules:

1. eliminate all equivalence signs by the rule $(\Phi \Leftrightarrow \Psi) \rightarrow (\Phi \Rightarrow \Psi) \wedge (\Psi \Rightarrow \Phi)$,

2. eliminate all implications by the rule $(\Phi \Rightarrow \Psi) \rightarrow (\neg\Phi \vee \Psi)$,

3. move all negation signs in front of the atoms: $(\neg\neg\Phi) \rightarrow \Phi$, $(\neg(\Phi \vee \Psi)) \rightarrow (\neg\Phi \wedge \neg\Psi)$, $(\neg(\Phi \wedge \Psi)) \rightarrow (\neg\Phi \vee \neg\Psi)$, $(\neg\forall x\ \Phi) \rightarrow (\exists x\ \neg\Phi)$, $(\neg\exists x\ \Phi) \rightarrow (\forall x\ \neg\Phi)$,

4. eliminate all existential quantifiers by the so called Skolemization operation: $(\exists x\Phi) \rightarrow (\Phi[x/f(y_1, \ldots, y_k)])$ where $y_1, \ldots, y_k$ are the free variables occurring in Φ, f is a newly generated *Skolem function*, and $\Phi[x/f(y_1, \ldots, y_k)]$ means replacing all occurrences of x by $f(y_1, \ldots, y_k)$ in Φ,

5. drop all universal quantifiers,

6. use the distributivity rules $((\Phi \vee \Psi) \wedge \Gamma) \rightarrow ((\Phi \wedge \Gamma) \vee (\Psi \wedge \Gamma))$ and $((\Phi \wedge \Psi) \vee \Gamma) \rightarrow ((\Phi \vee \Gamma) \wedge (\Psi \vee \Gamma))$ to move the $\wedge$ sign up to the top.

3.2 Resolution and Paramodulation

The resolution calculus, invented by John Alan Robinson [Rob65b] with the two rules *binary resolution* and *factorization* is a sound and *refutation complete* calculus operating on clauses: a PL1–formula is contradictory if and only if there is a sequence of resolutions and factorizations producing the *empty clause* as the elementary contradiction. This principle is used in many theorem provers for doing *refutation theorem proving* fully automatically.

The resolution rule takes two clauses with complementary literals and generates a new clause.

$$\begin{array}{ll} L, K_1, \ldots, K_k & \sigma \text{ is the most general unifier} \\ \neg L', M_1, \ldots, M_m & \text{of } L \text{ and } L' \\ \hline \sigma K_1, \ldots, \sigma K_k, \sigma M_1, \ldots, \sigma M_m & \end{array}$$

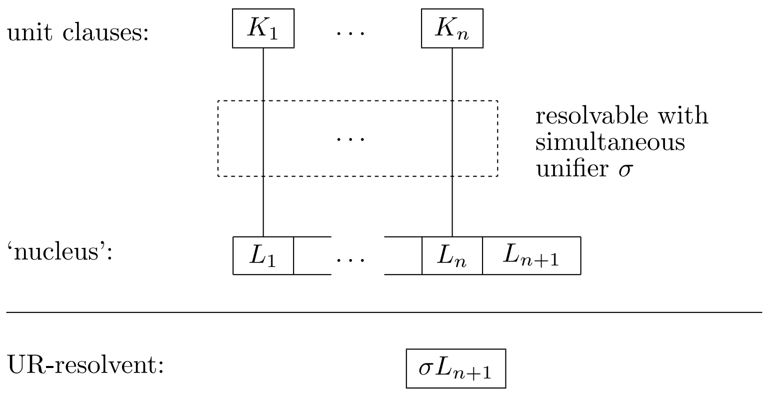
and the factorization rule generates a new clause from a clause with two 'unifiable' literals.

$$\begin{array}{ll} L, L', K_1, \ldots, K_k & \sigma \text{ is the most general unifier of } L \text{ and } L' \\ \hline \sigma L, \sigma K_1, \ldots, \sigma K_k & \end{array}$$

A *unifier* σ for two terms or literals L and L' is a substitution for the variables in L and L' that makes them equal. σ is *most general* if it instantiates as few variables as possible. For example, a most general unifier for $P(x, f(y))$ and $P(g(z), z)$ is $\{x \mapsto g(f(y)), z \mapsto f(y)\}$.

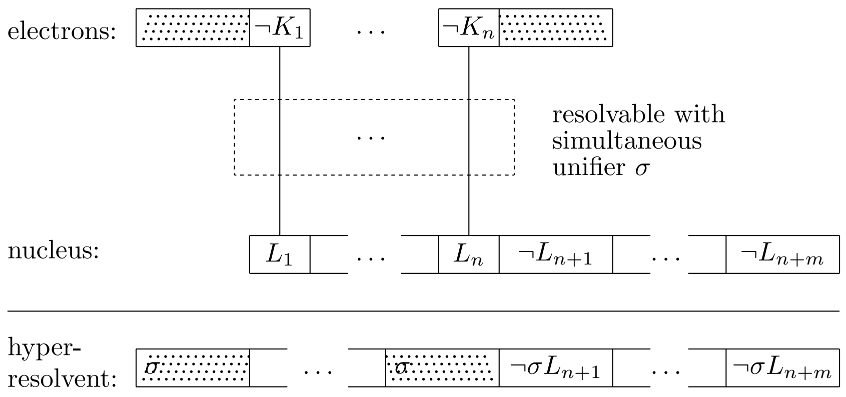
Various modifications and refinements of resolution have been developed. The most important ones for the purpose of this paper are *UR–resolution* ('Unit Resulting' resolution) and *Hyperresolution.* UR–resolution takes a clause with $n+1$ literals (the 'nucleus) and n *unit clauses* i.e. clauses with one literal, such that a sequence of resolutions would produce a new unit clause [MOW76]. This sequence of resolutions is done simultaneously.

The general schema for UR-resolution can be represented graphically as follows:



Boxes represent literals and a sequence of boxes represents a clause.

UR-resolution is not complete in general. It does not always find a proof if there exists one. An extension of UR-resolution, hyperresolution, however is complete [Rob65a]. Instead of unit clauses, hyperresolution takes as many clauses with *positive (negative) literals* only as the 'nucleus' has negative (positive) literals, and generates by simultaneous resolution a new positive (negative) clause. A graphical illustration is:



While UR- and hyperresolution have been invented to improve the search behaviour of theorem provers, the so called *paramodulation* rule has a different quality [RW69]. Paramodulation builds the meaning of the equality symbol directly into the calculus. In the simplest version, paramodulation takes an equation $a = b$ and a clause containing a and replaces the a by the b. If the terms contain variables and the equation is only one literal in a longer clause, an appropriate generalization of this idea is necessary. The general paramodulation rule is therefore:

$$\begin{array}{ll} L[t], K_1, \ldots, K_k & \sigma \text{ is the most general unifier} \\ l = r, M_1, \ldots, M_m & \text{of } l \text{ and a term occurrence } t \text{ in } L \\ \hline \sigma L[t \to r], \sigma K_1, \ldots, \sigma K_k, \sigma M_1, \ldots, \sigma M_m & L[t \to r] \text{means the occurrence of } t \text{ replaced by } r \end{array}$$

An example is

$$\begin{array}{ll} P(c, h(f(a,y),b)), \quad Q(y) & \sigma = \{x \mapsto a, y \mapsto d\} \\ f(x,d) = g(x), \quad R(x) & l \text{ is } f(x,d), \quad t \text{ is } f(a,y) \\ \hline P(c, h(g(a),b)), \quad Q(d), \quad R(a) & \end{array}$$

Paramodulation together with resolution and factorization are refutation complete for first order logic with equality. Special versions of paramodulation turned out to be very useful. The most important one is *demodulation* (or rewriting). In demodulation, the *demodulant* is a unit equation (unit clause with equality literal) that is applied almost like paramodulation to simplify new clauses as soon as possible. The differences to paramodulation are: (i) the demodulator is applied only from left to right, (ii) the unifier must be a 'matcher', i.e. it instantiates only the variables in the demodulator and (iii) paramodulation with the demodulator is done destructively, i.e. the old clause is deleted.

A typical application is: The demodulator is $x \times x^{-1} = 1$ (we use infix notation here). A clause $P(f(y) \times f(y)^{-1}), Q(y)$ would be demodulated to $P(1), Q(y)$ as soon as it appears in the search process, whereas a clause $P(f(y) \times f(a)^{-1}), Q(y)$ could not be demodulated because the y needs to be instantiated with a, and this is not allowed (paramodulation would do this).

Further notions which are important for clausal theorem proving are *tautology* and *subsumption.* A clause is a *tautology* if it contains a literal positive as well as negative. In refutation theorem proving (the goal is to derive a contradiction from the axioms together with the negated theorem), tautologies can (and should) be deleted as soon as possible. A clause A *subsumes* a clause B if an instance of A is a subset of B. For example $P(x)$ subsumes $P(a), Q$ because the '$x \mapsto a$'–instance of $P(x)$ is a subset of the second clause. Subsumed clauses can also be deleted without loosing refutation completeness. One distinguishes *forward subsumption* — a newly generated clause which is subsumed by an older one is deleted — and *backward subsumption* — all old clauses which are subsumed by a newly generated one are deleted.

Inference rules like resolution and paramodulation are nondeterministic. There are in general many alternatives in a clause set where they can be applied. Any implementation, however, has to apply them in a certain order. This gives the possibility to use selection heuristics and strategies for improving the performance of the system. *Ordering strategies* order the possible inference steps in a certain way, whereas *restriction strategies* forbid certain inferences completely. The restriction strategy which is important for our purposes is the so called set of support (SOS) strategy. The idea for this strategy came from the observation that resolutions between axiom clauses will never produce a contradiction — if the axioms are consistent. Therefore the SOS strategy starts with a distinguished subset of the initial clauses, the set of support and allows only inferences if at least one of the partners is in the set of support. The new clauses are put into the set of support again. The initial set of support has to be chosen by the user of the theorem prover. Typical candidates for the initial set of support are the negated theorem clauses.

4 A Framework for Automating Duality

Developing dualities as for example

(b) (i) $\forall X \subseteq U :\ X \subseteq F(X)$,
(ii) R is reflexive over U

from lemma 2.5 consists of four problems:

Top-Down Direction
1. Given (i), find a suitable candidate for (ii).
2. Verify the equivalence of (i) and (ii).

Bottom-Up Direction
3. Given (ii), find a suitable candidate for (i).
4. Verify the equivalence of (i) and (ii).

Up to now there was no method for solving the problems 1 and 3, except by pure guessing or by very special methods in certain limited cases (Sahlquist formulae in modal logic, for example [vB84]). Of course, people with experience in this quickly develop enough intuition for solving relatively simple problems of this kind. The more complex the formulae are, however, the less reliable is the intuition.

In contrast to this, our method is fully automatic and solves the guessing problem together with the verification problem in one go.

4.1 The Top–Down Direction

The top-down direction of the duality problem can be stated as follows: Given
(a) some functions F which are defined in terms of other relations and functions using the membership predicate $\in$:

(1) $Def(F, R)$: $\forall X_1, \ldots, X_n\ \forall x\ x \in F(X_1, \ldots, X_n) \Leftrightarrow \Phi$.

where Φ contains *no* occurrence of F, and
(b) a property $\Psi(F)$ of F that can be formulated in first–order predicate logic using again the special membership predicate $\in$, find a formula $\Gamma(R)$ such that

(2) $Def(F, R) \Rightarrow \Psi(F) \Leftrightarrow \Gamma(R)$.

In the case of power structures, $Def(F, R)$ is given by definition 2.1

(3) $\forall X_1, \ldots, X_n\ \forall x\ x \in F(X_1, \ldots, X_n)) \Leftrightarrow \exists x_1, \ldots, x_n\ x_1 \in X_1 \wedge \ldots \wedge x_n \in X_n \wedge R(x_1, \ldots, x_n, x)$

The version we need for the examples in lemma 2.5 is

(4) $\forall X\ \forall y\ y \in F(X) \Leftrightarrow \exists x\ x \in X \wedge R(x, y)$

Thus, the first requirement, a suitable predicate logic formulation for the definition of F is fulfilled in the case of power structures.

The second requirement is that the property of F has to be formulated as a predicate logic formula in terms of the $\in$ predicate. The structure of this formula $\Psi(F)$ must be such that application of $Def(F,R)$ as rewrite rule from left to right eliminates F completely and the resulting formula is of the structure

(5) $\Psi' = QX_1,\ldots,X_n\ \Psi''(\ldots \in X_1,\ldots,\ldots \in X_n)$ or equivalently
(6) $\Psi' = QX_1,\ldots,X_n\ \Psi''(X_1(\ldots),\ldots,X_n(\ldots))$

where Q is an existential or a universal quantifier. In the version (6), the set variables X_i have been replaced by their characteristic predicates. This brings to light the second–order nature of the problem which had been hidden in the membership predicate. Since Def is an equivalence, rewriting $\Psi(F)$ to $\Psi'(R)$ is an equivalence transformation in the theory of $Def(F,R)$, i.e. $Def(F,R) \Rightarrow \Psi(F) \Leftrightarrow \Psi'(R)$.

We illustrate this with the property (b)(i) of lemma 2.5: $\forall X \subseteq U :\ X \subseteq F(X)$. First of all, the property is reformulated in terms of the membership predicate

$$\forall X\ \forall y\ y \in X \Rightarrow y \in F(X).$$

The condition $X \subseteq U$ is obsolete because U denotes the whole domain of our interpretation. Now (4) is applied as rewrite rule and we get

$$\forall X\ \forall y\ y \in X \Rightarrow (\exists x\ x \in X \wedge R(x,y)).$$

or

$$\forall X\ \forall y\ X(y) \Rightarrow (\exists x\ X(x) \wedge R(x,y)).$$

respectively. The function F is eliminated now, but the resulting second–order formula is not yet satisfactory. What we are after is a first–order property in terms of the relation R. To this end, a formula $\Gamma(R)$ has to be found which is equivalent to $\Psi'(R)$, but does not contain the predicate variables X_i. This turned out to be the kernel of the problem. It can be solved by a quantifier elimination procedure that computes for a second–order formula an equivalent first–order formula — if there is one and the procedure succeeds. The particular quantifier elimination procedure we shall employ is discussed in some detail in the next section.

To summarize, the recipe for the top-down direction is:

1. Formulate the definition of the functions F in the style of (1).
2. Formulate the property $\Psi(F)$ in terms of the membership predicate.
3. Eliminate F from Ψ.
4. Replace the set variables X_i by their characteristic predicates.
5. Apply quantifier elimination.

4.2 The Bottom–Up Direction

In the bottom–up direction of the duality problem we want to compute from the property $\Gamma(R)$ of the relation R and the definition $Def(F,R)$ for the function F a corresponding property $\Psi(F)$. There are two different methods for computing Ψ. In the first method we exploit that

$$(\exists R\ \Gamma(R) \wedge Def(F,R)) \Leftrightarrow \Psi(F)$$

implies

$$Def(R,F) \Rightarrow (\Gamma(R) \Leftrightarrow \Psi(F)$$

for the particular Ψ and R. This reduces the problem again to a quantifier elimination problem. The quantifier $\exists R$ has to be eliminated from $\exists R\ \Gamma(R) \wedge Def(F,R))$. If this succeeds, we are done. Unfortunately it succeeds only in relatively simple cases. The reason is that both Def and Ψ are essentially second–order formulae. Only if this second–order nature is not relevant there is a chance with this method. An evidence for failure is that $\Gamma(R)$ is recursive, as for example transitivity.

The second method is much more complicated and it needs some heuristic guidance. It consists of a guessing and verification step. The guessing step, however, can be systematized such that the whole procedure is again fully automatic.

In the guessing step a theorem prover is used for synthesizing a candidate formula as a Skolem term. To this end, the connectives necessary to build $\Psi(F)$ as a term are axiomatized as function symbols and a formula

$$\exists f\ \forall x\ x \in f$$

is proved constructively. The binding $\Psi(F)$ of f used in the proof is the desired candidate formula. We enumerate the proofs and try to verify the generated formula with the top–down method. If there are enough connectives available the correct result should eventually be found.

Usually there are different options for the formulation of Ψ. If it can be expected that Ψ can be formulated in terms of the set connectives union, intersection, complement and subset, things are simpler. The axioms for these connectives are:

$$\begin{array}{ll}
\forall X, Y\ \forall x\ x \in \mathit{union}(X,Y) & \Leftrightarrow (x \in X \vee x \in Y) \\
\forall X, Y\ \forall x\ x \in \mathit{intersection}(X,Y) & \Leftrightarrow (x \in X \wedge x \in Y) \\
\forall X\ \forall x\ x \in \mathit{complement}(X) & \Leftrightarrow (\neg x \in X) \\
\forall X, Y\ \forall x\ x \in \mathit{subset}(X,Y) & \Leftrightarrow (x \in X \Rightarrow x \in Y)
\end{array}$$

The *subset* connective is actually an abbreviation: $\mathit{subset}(X,Y) = \mathit{complement}(X) \cup Y$ and can be used to model the normal subset relation as a function.

The input to the theorem prover consists of these axioms, together with $\mathit{Def}(F,R)$ and $\Gamma(R)$. The theorem to be proven is $\exists f\ \forall x\ x \in f$. The result are proofs with bindings for f, for example $f = \mathit{subset}(F(X), F(F(X)))$, i.e. $F(X) \subseteq F(F(X))$.

Things get more complicated if the formula Ψ contains special predicates on sets and logical connectives. The correspondence

(i) (i) F maps singletons onto singletons,
(ii) R is a unary operation over U

of lemma lemma 2.5 is such a case. In order to synthesize 'F maps singletons onto singletons' we must axiomatize a predicate *singleton*, the connective *implies* and synthesize the term

$$\mathit{implies}(\mathit{singleton}(X), \mathit{singleton}(F(X))).$$

That means in particular that *singleton* must be defined as a function. This is only possible by means of a *Holds*–predicate:

$$\forall X\ \mathit{Holds}(\mathit{singleton}(X)) \Leftrightarrow \forall x, y\ x \in X \wedge y \in X \Leftrightarrow x = y$$

The axiom for the implication as function is

$$\forall X\ \mathit{Holds}(\mathit{implies}(X,Y)) \Leftrightarrow \mathit{Holds}(X) \Rightarrow \mathit{Holds}(Y).$$

From axioms of this kind we could try to prove $\exists f \mathit{Holds}(f)$. Unfortunately this approach turned out to be intractable for technical reasons. Since automated theorem provers usually negate the theorem and search for a refutation, the negation of $\exists f\ \mathit{Holds}(f)$ which is $\forall f \neg \mathit{Holds}(f)$ is added to the formula set. This causes the problem that all formulae with negated occurrence of *Holds* get subsumed and deleted. The problem becomes unsolvable. Turning subsumption off is no solution because the search space gets so terribly large that no interesting theorem can be proven.

A much more elegant solution comes from the possible worlds idea in modal logic. We make the *Holds*–predicate and the $\in$–relation world dependent. That means we use the definitions

$$\begin{array}{l}
\forall w\ \forall X\ \mathit{Holds}(w, \mathit{singleton}(X)) \Leftrightarrow \forall x, y\ \in (w, x, X) \wedge \in (w, y, X) \Leftrightarrow x = y \\
\forall w\ \forall X\ \mathit{Holds}(w, \mathit{implies}(X,Y)) \Leftrightarrow \mathit{Holds}(w, X) \Rightarrow \mathit{Holds}(w, Y).
\end{array}$$

and prove a theorem $\exists f\ \forall w\ Holds(w, f)$. From a logical point of view, this 'world'–argument is redundant, but it avoids the subsumption problem. The negated theorem $\forall f \exists w\ \neg Holds(w, f)$ gets Skolemized to $\neg Holds(k(w), f)$ and does not subsume anything.

Summarizing, we propose the following procedure for computing $\Psi(F)$ from $Def(F, R)$ and $\Gamma(R)$:

1. Try quantifier elimination for $\exists R\ Def(R, F) \wedge \Gamma(R)$.
   If this does not succeed:
2. Try to find a solution in terms of set connectives.
   (a) Axiomatize the set connectives.
   (b) From these axioms together with $Def(F, R)$ and $\Gamma(R)$ prove the theorem
   $\exists f\ \forall x\ x \in f$.
   (c) Each binding for f is a candidate for $\Psi(F)$ that needs to be verified with the top–down method.
3. Try to find a solution in terms of general connectives and predicates on sets.
   (a) Axiomatize the connectives and the predicates on sets (for example 'singleton', 'emptyset' etc.) with a world dependent *Holds*–predicate
   (b) From these axioms together with $Def(F, R)$ and $\Gamma(R)$ prove the theorem
   $\exists f\ \forall w\ Holds(w, f)$.
   (c) Each binding for f is a candidate for $\Psi(F)$ that needs to be verified with the top–down method.

The procedures will be illustrated in detail in the examples section.

5 Quantifier Elimination

In [GO92] we have developed an algorithm which can compute for second–order formulae of the kind $\exists P_1, \ldots, P_k\ \Phi$ where Φ is a first–order formula, an equivalent first–order formula — if there is one. Since $\forall P_1, \ldots, P_k\ \Phi \Leftrightarrow \neg\exists P_1, \ldots, P_k\ \neg\Phi$ this algorithm can also be applied to our case. Related methods can also be found in [Ack35a, Ack35b, Ack54, Sza92, BGW92, Sim93].

The definition of the algorithm is:

Definition 5.1 (The SCAN Algorithm)
Input to SCAN is a formula $\alpha = \exists P_1, \ldots, P_n\ \psi$ with predicate variables $P_1, \ldots, P_n$ and an arbitrary first–order formula ψ.
Output of the SCAN — if it terminates — is a formula φ_α which is *logically equivalent* to α, but not containing the predicate variables $P_1, \ldots, P_n$.
SCAN performs the following three steps:

1. ψ is transformed into clause form.
2. All C–resolvents and C–factors with the predicate variables $P_1, \ldots, P_n$ have to be generated. C–resolution ('C' for constraint) is defined as follows:

$$\begin{array}{ll} P(s_1, \ldots, s_n) \vee C & P(\ldots) \text{ and } \neg P(\ldots) \\ \neg P(t_1, \ldots, t_n) \vee D & \text{are the } \textit{resolution literals} \\ \hline C \vee D \vee s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n & \end{array}$$

   and the C-factorization rule is defined analogously:

$$\frac{P(s_1, \ldots, s_n) \vee P(t_1, \ldots, t_n) \vee C}{P(s_1, \ldots, s_n) \vee C \vee s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n}.$$

Notice that only C-resolutions between different clauses are allowed (no self resolution). A C-resolution or C-factorization can be optimized by destructively resolving literals $x \neq t$ where the variable x does not occur in t with the reflexivity equation. C–resolution and C–factorization takes into account that second order quantifiers may well impose conditions on the interpretations which must be formulated in terms of equations and inequations.

As soon as *all* resolvents and factors between a particular literal and the rest of the clause set have been generated (the literal is 'resolved away'), the clause containing this literal must be deleted (purity deletion). If all clauses are deleted this way, this means that α is a tautology.

All equivalence preserving simplifications may be applied freely. These are for example:

- Tautologous resolvents can be deleted.
- Subsumed clauses can be deleted.
- Subsumption factoring can be performed. Subsumption factoring means that a factor subsumes its parent clause. This may be realized by just deleting some literals. For example $Q(x), Q(a)$, where x is a variable, can be simplified to $Q(a)$.
- Subsumption resolution can also be performed. Subsumption resolution means that a resolvent subsumes its parent clause, and this again may be realized by deleting some literals [OS91]. For example the resolvent between P, Q and $\neg P, Q, R$ is just Q, R such that $\neg P$ can be deleted from the clause.

If an empty clause is generated, this means that α is contradictory.

3. If the previous step terminates and there are still clauses left then reverse the skolemization. If this is not possible, the only chance is to take parallel (second–order) Henkin quantifiers [Hen61].

$\lhd$

The next example illustrates the different steps of the SCAN algorithm in more detail. The input is: $\exists P\ \forall x, y\ \exists z\ (\neg P(a) \vee Q(x)) \wedge (P(y) \vee Q(a)) \wedge P(z)$.
In the first step the clause form is to be computed:

$$\begin{array}{ll} C_1 & \neg P(a), Q(x)) \\ C_2 & \quad P(y), Q(a) \\ C_3 & \quad P(f(x,y)) \end{array}$$

f is a Skolem function.

In the second step of SCAN we begin by choosing $\neg P(a)$ to be resolved away. The resolvent between C_1 and C_2 is $C_4 = Q(x), Q(a)$ which is equivalent to $Q(a)$ (this is one of the equivalence preserving simplifications). The C-resolvent between C_1 and C_3 is $C_5 = (a \neq f(x,y), Q(x))$. There are no more resolvents with $\neg P(a)$. Therefore C_1 is deleted. We are left with the clauses

$$\begin{array}{ll} C_2 & P(y), Q(a) \\ C_3 & P(f(x,y)) \\ C_4 & Q(a) \\ C_5 & a \neq f(x,y), Q(x) \end{array} .$$

Selecting the next two P-literals to be resolved away yields no new resolvents. Thus, C_2 and C_3 are simply to be deleted as well. All P-literals have now been eliminated. Restoring the quantifiers we then get

$$\forall x\ \exists z\ Q(a) \wedge (a \neq z \vee Q(x))$$

as the final result.

The SCAN algorithm is correct in the sense that its result is really equivalent to the input formula. It cannot be complete, i.e. there may be second–order formulae which have a first–order equivalent, but SCAN cannot find it. This is not possible in general, otherwise the theory of arithmetic would be enumerable.

The points where SCAN can fail are (i) the resolution does not terminate and (ii) reversing Skolemization is not possible. In the second case there is a (again second–order) solution in terms of parallel Henkin quantifiers.

6 The Theorem Prover OTTER

Among the currently available tools which can be used for our purposes, the theorem prover OTTER, developed by Bill McCune, turned out to be the most appropriate one. We give a brief description of those features of OTTER which are relevant for our purposes. Most of it is taken from the OTTER manual. Those features which are not relevant for this paper are omitted. The reader who wants to experiment with OTTER can find the details in the OTTER manual [McC89] or in [WOLB91].

OTTER (Organized Techniques for Theorem-proving and Effective Research) is a resolution-style theorem prover, similar in scope and purpose to the AURA [Smi88] and LMA/ITP [LO84] theorem provers, all of them developed at Argonne National Laboratory. The primary design considerations have been performance, portability, and compactness and simplicity of the code. The programming language C is used.

OTTER features the inference rules binary resolution, hyperresolution, UR-resolution, and binary paramodulation. These inference rules take a small set of clauses and infer a clause; if the inferred clause is new, interesting, and useful, it is stored and may become available for subsequent inferences.

Other features of OTTER are the following:

- Statements of the problem may be input either with first-order formulas or with clauses (a clause is a disjunction with implicit universal quantifiers and no existential quantifiers). If first-order formulas are input, OTTER translates them to clauses.
- Forward demodulation rewrites and simplifies newly inferred clauses with a set of equalities, and back demodulation uses a newly inferred equality (which has been added to the set of demodulators) to rewrite all existing clauses.
- Forward subsumption deletes an inferred clause if it is subsumed by any existing clause, and back subsumption deletes all clauses that are subsumed by an inferred clause.
- Weight functions and lexical ordering decide the "goodness" of clauses and terms.
- Answer literals give information about the proofs that are found.

OTTER is not automatic. Even after the user has encoded a problem into first-order logic or into clauses, the user must choose inference rules, set options to control the processing of inferred clauses, and decide which input formulas or clauses are to be in the initial set of support and which (if any) equalities are to be demodulators. If OTTER fails to find a proof, the user may wish to try again with different initial conditions.

OTTER uses the given-clause algorithm, which can be viewed as a simple implementation of the set of support strategy. OTTER maintains three lists of clauses: `axioms`, `sos` (set of support), and `demodulators`. OTTER appends clauses that have been given to `usable` rather than keeping them in a separate list.
The main loop for inferring and processing clauses and searching for a refutation is

```
While (sos is not empty and no refutation has been found)
    1. Let given_clause be the ‘lightest’ clause in sos;
    2. Move given_clause from sos to usable;
    3. Infer and process new clauses using the inference rules in
           effect; each new clause must have the given_clause as
           one of its parents and members of usable as its other
```

```
            parents;  new clauses that pass the retention tests
            are appended to sos;
    End of while loop.
```

The procedure for processing a newly inferred clause `new_cl` is

```
 1.  (optional) Output new_cl.
 2.  Demodulate new_cl.
 3.  (optional) Orient equalities.
 4.  Merge identical literals (leftmost copy is kept).
 5.  (optional) Sort literals.
 6.  (optional) Discard new_cl and exit if new_cl has too many literals.
 7.  Discard new_cl and exit if new_cl is a tautology.
 8.  (optional) Discard new_cl and exit if new_cl is too ‘heavy’.
 9.  (optional) Discard new_cl and exit if new_cl is subsumed by any clause
         in axioms or sos (forward subsumption).
10.  (optional) Apply unit deletion.
11.  Integrate new_cl and append it to sos.
12.  (optional) Output kept clause.
13.  If new_cl has 0 literals, a refutation has been found.
14.  If new_cl has 1 literal, then search axioms and sos for
         unit conflict (refutation) with new_cl.
15.  (optional) Print the proof if a refutation has been found.
16.  (optional) Try to make new_cl into a demodulator.
17.  (optional) Back demodulate if Step 16 made new_cl into a demodulator.
18.  (optional) Discard each clause in axioms and each clause in sos that
         is subsumed by new_cl (back subsumption).
19.  (optional) Factor new_cl and process factors.
```

Steps 17–19 are delayed until steps 1–16 have been applied to all clauses inferred from the current given clause.

Input to OTTER consists of a small set of commands, some of which indicate that a list of objects (clauses, formulas, or weight templates) follows the command. All lists of objects are terminated with `end_of_list`.

```
set(flag_name).                   % set a flag
clear(flag_name).                 % clear a flag
assign(parameter_name,integer).   % assign an integer to a parameter
list(usable).                     % read axioms in clause form
list(sos).                        % read set of support in clause form
list(demodulators).               % read demodulators in clause form
formula_list(usable).             % read axioms in formula form
formula_list(sos).                % read set of support in formula form
weight_list(weight_list_name).    % read weight templates
```

OTTER recognizes two kinds of options: flags and parameters. Flags are Boolean-valued options; they are changed with the `set` and the `clear` commands, which take the name of the flag as the argument. Parameters are integer-valued options; they are changed with the `assign` command, which takes the name of the parameter as the first argument and an integer as the second. Examples are

```
set(binary_res).                 % switch on binary resolution
clear(back_sub).                 % do not use back subsumption
assign(max_proofs, 300).         % stop after the 300th proof
assign(max_weight, 100).         % delete all clauses ‘heavier’ than 100.
```

Answer Literals

The main use of answer literals is to record, during a search for a refutation, instantiations of variables in input clauses. For example, if the theorem under consideration states that an object exists, then the denial of the theorem contains a variable, and an answer literal containing the variable can be appended to the denial. If a refutation is found, then the empty clause has an answer literal that contains the object whose existence has just been proved.

Any literal whose predicate symbol starts with `$ans`, `$Ans`, or `$ANS` is an answer literal. Most routines—including the ones that count literals and decide whether a clause is positive or negative—ignore any answer literals. The inference rules insert, into the children, the appropriate instances of any answer literals in the parents. If factoring is enabled, OTTER *does* attempt to factor answer literals.

Weighting and Demodulation

The details of the input syntax should be self explaining in the examples ('|' is used for $\vee$). In our application, we need, however, a combination of weighting and demodulation which needs to be explained in more detail. Suppose you want to get rid of clauses which contain terms of the kind `F(F(...))`, i.e. double nesting of F should be avoided. This can be achieved simply by specifying a weighting template

```
weight_list(purge_gen).
        weight(F(F(*1)),200).
end_of_list.
```

and setting the maximum weight, say to 100. Clauses containing such terms become 'too heavy' and are deleted. (`*n` in the weighting templates means that the weight of the subterm is multiplied with n. The weight of the term is then added to the second argument of the `weight` specification to get the final weight.)

Things get more complicated, if clauses with particular terms are to be deleted, for example clauses with terms `s(p,s(q,p))` where `p` and `q` are arbitrary terms, but the two occurrences of `p` must be identical. These clauses can be deleted by demodulating them to some term `junk` and making the term `junk` too heavy:

```
list(demodulators).
        (s(x,s(y,x)) = junk).
end_of_list.
weight_list(purge_gen).
        weight(junk,200).
end_of_list.
```

Finally, in our application we avoid 'junk proofs' by rewriting `$ans`–literals to `$T` which stands for truth and causes the clauses to be deleted as tautologies. For example

```
list(demodulators).
        ($ans(F(s(x,y))) = $T).
end_of_list.
```

causes all clauses with this pattern to be deleted immediately.

7 Strategies and Heuristics for the Bottom–Up Direction

The purpose of this paper is not only to explain the methods in general terms, but also to demonstrate that it really works. Moreover we want to enable the reader who has access to a theorem prover like OTTER to experiment himself. A naive approach to such experiments, however, will definitely result in frustration. To avoid frustration we have to give a number of more or less technical hints.

Let us briefly repeat the method for the bottom–up direction

In order to compute $\Psi(F)$ from $Def(F, R)$ and $\Gamma(R)$:

1. Try quantifier elimination for $\exists R\ Def(R, F) \wedge \Gamma(R)$.
   If this does not succeed:

2. Try to find a solution in terms of set connectives.

   (a) Axiomatize the set connectives:

   $$\begin{array}{ll}\forall X,Y\ \forall x\ x \in \mathit{union}(X,Y) & \Leftrightarrow (x \in X \vee x \in Y)\\ \forall X,Y\ \forall x\ x \in \mathit{intersection}(X,Y) & \Leftrightarrow (x \in X \wedge x \in Y)\\ \forall X\ \forall x\ x \in \mathit{complement}(X) & \Leftrightarrow (\neg x \in X)\\ \forall X,Y\ \forall x\ x \in \mathit{subset}(X,Y) & \Leftrightarrow (x \in X \Rightarrow x \in Y)\end{array}$$

   (b) From these axioms together with $\mathit{Def}(F,R)$ and $\Gamma(R)$ prove the theorem $\exists f\ \forall x\ x \in f$.

   (c) Each binding for f is a candidate for $\Psi(F)$ that needs to be verified with the top–down method.

3. Try to find a solution in terms of general connectives and predicates on sets.

   (a) Axiomatize the connectives and the predicates on sets with world dependent *Holds*–predicate and world dependent $\in$–predicate:

   $$\begin{array}{ll}\forall w\ \forall X,Y\ \mathit{Holds}(w,\mathit{or}(X,Y)) & \Leftrightarrow (\mathit{Holds}(w,X) \vee \mathit{Holds}(w,Y))\\ \forall w\ \forall X,Y\ \mathit{Holds}(w,\mathit{and}(X,Y)) & \Leftrightarrow (\mathit{Holds}(w,X) \wedge \mathit{Holds}(w,Y))\\ \forall w\ \forall X\ \mathit{Holds}(w,\mathit{not}(X)) & \Leftrightarrow (\neg \mathit{Holds}(w,X)\\ \forall w\ \forall X\ \mathit{Holds}(w,\mathit{implies}(X,Y)) & \Leftrightarrow \mathit{Holds}(w,X) \Rightarrow \mathit{Holds}(w,Y).\end{array}$$

   $$\begin{array}{ll}\forall w\ \forall X\ \mathit{Holds}(w,\mathit{singleton}(X)) & \Leftrightarrow \forall x,y\ \in(w,x,X) \wedge \in(w,y,X) \Leftrightarrow x = y\\ \forall w\ \forall X\ \mathit{Holds}(w,\mathit{empty}(X)) & \Leftrightarrow \neg\exists x\ \in(w,x,X)\\ \forall w\ \forall X\ \mathit{Holds}(w,\mathit{notempty}(X)) & \Leftrightarrow \exists x\ \in(w,x,X).\end{array}$$

   The axioms for the set connectives, formulated with world dependent $\in$–predicate may also be necessary in general. For the examples in this paper they are not necessary.

   (b) From these axioms together with $\mathit{Def}(F,R)$ and $\Gamma(R)$ prove the theorems $\exists f\ \forall w\ \mathit{Holds}(w,f)$.

   (c) Each binding for f is a candidate for $\Psi(F)$ that needs to be verified with the top–down method.

The first trial, quanitifier elimination of $\exists R\ \mathit{Def}(R,F) \wedge \Gamma(R)$ succeeds only in a few simple cases. In the more complicated cases we have to apply the guess and verify method. The guessing part, which amounts to enumerating proofs, requires some guidance to restrict the search space.

7.1 General Options for Restricting the Search Space

In the guessing part, a theorem prover synthesizes logical formulae as terms. If logical connectives and predicates are encoded as function symbols, the theorem prover does not know about the theory behind these symbols (boolean algebra). In our case, this does not cause any incompleteness because the connectives are axiomatized properly. But it causes a lot of redundancy and useless steps. There are two things one can do against this phenomena.

1. Use a sort mechanism with sorts *Set* and *Bool* to distinguish between predicates, logical connectives and set variables. Typical sort declarations are:

   $$\begin{array}{ll}\mathit{singleton}: & \mathit{Set} \to \mathit{Bool}\\ \mathit{implies}: & \mathit{Bool} \times \mathit{Bool} \to \mathit{Bool}\\ \mathit{union}: & \mathit{Set} \times \mathit{Set} \to \mathit{Set}\\ F: & \mathit{Set} \to \mathit{Set}\end{array}$$

   This avoids the generation of nonsense terms like $\mathit{singleton}(\mathit{singleton}(X))$.

   If the theorem prover does not support sorts — and this is the case for OTTER — there are two possibilities to simulate the effect of sorts:

(a) Injection Functions
If there are no two sorts with common subsorts in the sort hierarchy, one can represent a term s of sort B as a nested term $A_1(\ldots A_n(B(s))\ldots)$ where $A_1 \ldots A_n B$ is the path in the sort hierarchy from the top sort A_1 to the actual sort B. This trick makes the unsorted unification behave like sorted unification. In our case there are only the sorts *Set* and *Bool*. Terms s of sort *Set* can be represented simply as $Set(s)$ and terms t of sort *Bool* can be represented as $Bool(t)$. For example the definition of the *singleton*–predicate would be

$$\begin{array}{l}\forall w\ \forall X\ Holds(w, Bool(singleton(Set(X)))) \\ \quad \Leftrightarrow \forall x, y\ \in(w, x, Set(X)) \wedge \in(w, y, Set(X)) \Leftrightarrow x = y.\end{array}$$

(b) Unfortunately injection functions make the formulae almost unreadable Therefore we chose a simpler method which is sufficient for our examples. Using a demodulator

$$(singleton(singleton(x)) = junk).$$

clauses with unwanted terms of this kind are deleted as soon as they are generated.

2. Ideally one would like to have the unification theory for boolean algebras for the logical connectives and the set connectives in order to enable subsumption modulo this theory. If this theory is not available, and this is usually the case, we must simulate the effect using demodulation. The idea is: Each clause containing a term that can be simplified by pure logical means, in particular those simplifying to *true*, can be deleted. The reason is that our axiomatization ensures that the simplified version eventually gets generated anyway. Typical demodulators are:

$$\begin{array}{ll}(complement(complement(x)) & = junk). \\ (implies(x, x) & = junk). \\ (implies(x, implies(y, x)) & = junk). \\ (subset(x, subset(y, x)) & = junk). \\ \ldots & \end{array}$$

7.2 The Ideal Search Procedure

The main parameter responsible for the size of the search space is the language, i.e. the set of connectives we provide for formulating $\Psi(F)$. In most cases of our example set, the subset connective is sufficient. If in addition to this, union, intersection, complement etc. are axiomatized, the search space is blown up for nothing. But the appropriate set of connectives is unknown in general. Therefore the first heuristic of the general procedure is: Start several search processes in parallel on different processors, each one with a different subset of the available connectives. In networks of workstations, this should be no problem. Since we knew the solution for our examples below in advance, we tried only the relevant subset of connectives and predicates.

Even with a limited number of connectives and predicates, we have to control the enumeration of the proofs (even with a tight control, almost 2000 proofs had to be generated for the example a) with ternary relations below before the right one appeared.) One way to control the enumeration of proofs is by limiting the size of the terms during the search and increasing the limit step by step. In the ideal search procedure, this might be the simplest way to do it. With the OTTER theorem prover it turned out to be more effective if certain nesting of terms are restricted. For example no case occurred so far where nested *subset* connectives were needed. Limiting the direct nesting of set connectives and logical connectives and increasing this limit step by step seems to work quite well.

In our example cases we applied the following heuristics:

1. No connectives inside the F–function are allowed.

2. No nested set connectives, logical connectives and set predicates are allowed. Of course, set predicates like *singleton* may occur inside logical connectives like *implies*. But terms like $implies(\ldots, implies(\ldots, \ldots))$ were not allowed.

3. Control the nesting of the F–function

The restrictions 1 and 2 are the first level of the search. It is sufficient for the examples. Only F needs to be nested more than once in some cases.

A reasonable and fair policy for enumerating the allowed syntactic structure for $\Psi(F)$, however, has yet to be worked out.

8 Examples

We go through the examples of lemma 2.5 and lemma 2.6 one by one and show the top–down and bottom-up solutions together with the technical details necessary to repeat the experiment with the Otter theorem prover. The list of examples was compiled (in [Bri89]) before our method had been developed. Thus, this is *not* a selection of cases where our method just happens to work.

Let us briefly repeat the recipe for the top–down direction
1. Formulate the definition of the functions F in the style of (1) (page 10).
2. Formulate the property $\Psi(F)$ in terms of the membership predicate.
3. Eliminate F from Ψ.
4. Replace the set variables by their characteristic predicates.
5. Apply quantifier elimination. The SCAN algorithm proceeds as follows:
6. If the formula is universally quantified: Negate it.
7. Generate a clause form.
8. 'Resolve away' all predicates to be eliminated and delete the pure clauses afterwards.
9. Reconstruct the quantifiers.
10. If the original formula had been negated, negate the result again.

The only example in our list which requires more than two or three resolution steps is example (f) for ternary relations. We use this example to demonstrate the use of the Otter theorem prover for performing quantifier elimination. In all the other examples quantifier elimination is done by hand.

The bottom–up direction has been discussed in detail in section 7. For lack of a particular optimized version for this direction we use Otter as it is. The control parameters for Otter which are the same for all examples are:

```
set(unit_deletion).
assign(max_proofs,2000).
assign(max_weight,100).
set(para_from).
```

All other parameters are listed for the examples separately. The following abbreviations are used for the connectives and predicates:

e	= membership relation $\in$
H	= *Holds*–predicate
ep	= emptyset–predicate $\emptyset$
ne	= not emptyset–predicate $\neg\emptyset$
sg	= singleton predicate
s	= subset connective $\subseteq$
i	= implication $\Rightarrow$.

The naming convention for variable symbols and constant symbols is: Variable symbols are taken from the end of the alphabet u, v, w, x, y, z. Constant symbols which are generated from existential quantifiers are written underlined. That means for example $\underline{x}$ is a Skolem constant stemming from a '$\exists x$' quantification.

Weightings and demodulators like

```
        (i(x,x) = junk).
        ($ans(i(i(x,y),z)) = $T).
        ($ans(i(x,i(y,z))) = $T).
        ($ans(i(x,x)) = $T).
        ($ans(junk) = $T).
```

prevent the generation of nested connectives. (In a fully automated procedure this would be gradually relaxed.)

The CPU times for most of the examples range from a few seconds to a few minutes on a Macintosh IIfx. Only for example (a) for ternary relations, more than 1 hour was needed on a Solburn machine.

8.1 Binary Relations

We begin with the examples of lemma 2.5. They correlate a binary relation R with a unary function F. The definition of F for this case (def. 2.1) is used as a rewrite rule:

(r1) $(y \in F(X)) \to (\exists x\ x \in X \wedge R(x,y))$

Example a

(a) (i) $\forall X \subseteq U :\ X \neq \emptyset \Rightarrow F(X) \neq \emptyset$,

(ii) Domain $(R) = U$

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$(\exists x\ x \in X) \Rightarrow (\exists y\ y \in F(X))$
Rewritten:	$(\exists x\ X(x)) \Rightarrow (\exists y\ (\exists x\ X(x) \wedge R(x,y)))$
Negated and Clausified:	$X(\underline{x})$
	$\neg X(x), \neg R(x,y)$
X 'Resolved Away':	$\neg R(\underline{x},y)$
Quantifiers Reconstructed:	$\exists x\ \forall y\ \neg R(x,y)$
Negated again:	$\forall x\ \exists y\ R(x,y)$ (which means $domain(R) = U$)

Direction (ii) → (i) (Bottom-Up)

First trial: Quantifier Elimination

$\exists R\ \Gamma(R) \wedge Def(F,R) =$

$\exists R\ (\forall x\ \exists y\ R(x,y)) \wedge (\forall y\ e(y,F(X)) \Leftrightarrow \exists x\ e(x,X) \wedge R(x,y))$

Clauses:

$R(x,f(x))$

$\neg e(y,F(X)), e(g(y),X)$

$\neg e(y,F(X)), R(g(y),y)$

$\neg e(x,X), \neg R(x,y), e(y,F(X))$

R resolved away

$\neg e(y,F(X)), e(g(y),X)$

$\neg e(x,X), e(f(x),F(X))$

Quantifiers reconstructed:

$\forall y\ e(y,F(X)) \Rightarrow \exists z\ e(z,X)$

$\forall x\ e(x,X) \Rightarrow \exists y\ e(y,F(X))$

The set notation of the combined formula is:

$(\exists x\ x \in X \Leftrightarrow (\exists y\ y \in F(X))$ which implies $X \neq \emptyset \Rightarrow F(X) \neq \emptyset$

Although the quantifier elimination trial was successful, we also show the theorem prover version. It illustrates the use of the world dependent *Holds*–predicate.

OTTER **Protocol:**

```
set(binary_res).

formula_list(usable).
(all w (all z (all X (e(w,z,F(X)) <-> (exists x (e(w,x,X) & R(x,z))))))).
(all w (all X (all Y (H(w,i(X,Y)) <-> (H(w,X) -> H(w,Y)))))).
(all w (all X (H(w,ne(X)) <-> (exists x e(w,x,X))))).
end_of_list.

formula_list(sos).
(all x (exists y R(x,y))).
-(exists f (all w (H(w,f) & -$ans(f)))).
end_of_list.

Clauses:

list(usable).
7  -e(w,z,F(x1)) | e(w,$f1(w,z,x1),x1).     8  -e(w,z,F(x1)) | R($f1(w,z,x1),z).
9  e(w,z,F(x1)) | -e(w,x,x1) | -R(x,z).     10 -H(w,i(x2,x3)) | -H(w,x2) | H(w,x3).
11 H(w,i(x2,x3)) | H(w,x2).                  12 H(w,i(x2,x3)) | -H(w,x3).
13 -H(w,ne(x4)) | e(w,$f2(w,x4),x4).         14 H(w,ne(x4)) | -e(w,x,x4).
end_of_list.

list(sos).
15  R(x,$f3(x)).
16  -H($f4(x5),x5) | $ans(x5).
end_of_list.

% Heuristic: Avoid double nestings of i,F,ne.
weight(i(i(*1,*1),*1),200).     weight(i(*1,i(*1,*1)),200).
weight(F(F(*1)),200).           weight(ne(ne(*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (i(x,x) = junk).             2  ($ans(i(i(x,y),z)) = $T).
3  ($ans(i(x,i(y,z))) = $T).    4  ($ans(i(x,x)) = $T).
6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
9  e(x,y,F(z)) | -e(x,u,z) | -R(u,y).
11  H(x,i(y,z)) | H(x,y).
12  H(x,i(y,z)) | -H(x,z).
13  -H(x,ne(y)) | e(x,$f2(x,y),y).
14  H(x,ne(y)) | -e(x,z,y).
15  R(x,$f3(x)).
16  -H($f4(x),x) | $ans(x).
17 [binary,15,9] e(x,$f3(y),F(z)) | -e(x,y,z).
19 [binary,16,12] $ans(i(x,y)) | -H($f4(i(x,y)),y).
21 [binary,16,11] $ans(i(x,y)) | H($f4(i(x,y)),x).
26 [binary,19,14] $ans(i(x,ne(y))) | -e($f4(i(x,ne(y))),z,y).
30 [binary,21,13] $ans(i(ne(x),y)) | e($f4(i(ne(x),y)),$f2($f4(i(ne(x),y)),x),x).
48 [binary,17,26] -e($f4(i(x,ne(F(y)))),z,y) | $ans(i(x,ne(F(y)))).
```

```
49 [binary,48,30] $ans(i(ne(v64),ne(F(v64)))).
------------ end of proof -------------
```

In set notation, the answer is $\forall X\ X \neq \emptyset \Rightarrow F(X) \neq \emptyset$

Example b

(b) (i) $\forall X \subseteq U :\ X \subseteq F(X)$,
(ii) R is reflexive over U

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$\forall y\ y \in X \Rightarrow y \in F(X)$
Rewritten:	$\forall y\ X(y) \Rightarrow (\exists x\ X(x) \wedge R(x,y))$
Negated and Clausified:	$X(\underline{y})$
	$\neg X(x), \neg R(x, \underline{y})$
X 'Resolved Away':	$\neg R(\underline{y}, \underline{y})$
Quantifiers Reconstructed:	$\exists y\ \neg R(y,y)$
Negated again:	$\forall y\ R(y,y)$

Direction (ii) → (i) (Bottom-Up)
First trial: Quantifier Elimination

$\exists R\ \Gamma(R) \wedge Def(F,R) =$
$\exists R\ (\forall x\ R(x,x)) \wedge (\forall y\ e(y,F(X)) \Leftrightarrow \exists x\ e(x,X) \wedge R(x,y))$

Clauses:

$R(x,x)$
$\neg e(y,F(X)), e(g(y),X)$
$\neg e(y,F(X)), R(g(y),y)$
$\neg e(x,X), \neg R(x,y), e(y,F(X))$

R resolved away

$\neg e(y,F(X)), e(g(y),X)$
$\neg e(x,X), e(x,F(X))$

Quantifiers reconstructed:

$\forall y\ e(y,F(X)) \Rightarrow \exists x\ e(x,X)$
$\forall x\ e(x,X) \Rightarrow e(x,F(X)$

The logical notation of the second formula (the first one is true anyway) is:

$\forall x\ x \in X \Rightarrow x \in F(X)$ which means nothing else than $X \subseteq F(X)$

Again we list the result of the guess and verify procedure.

OTTER **Protocol**

```
set(binary_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
end_of_list.

formula_list(sos).
(all x R(x,x)).
-(exists f (all z (e(z,f) & -$ans(f)))).
```

```
end_of_list.

Clauses:

list(usable).
6 -e(z,F(x1)) | e($f1(z,x1),x1).        7 -e(z,F(x1)) | R($f1(z,x1),z).
8 e(z,F(x1)) | -e(x,x1) | -R(x,z).      9 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
10 e(z,s(x2,x3)) | e(z,x2).            11 e(z,s(x2,x3)) | -e(z,x3).
end_of_list.

list(sos).
12 R(x,x).                             13 -e($f2(x4),x4) | $ans(x4).
end_of_list.

% Heuristic: Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).             weight(s(*1,s(*1,*1)),200).
weight(F(F(*1)),200).                   weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).                     2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).            4  ($ans(s(x,x)) = $T).
5  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
8  e(x,F(y)) | -e(z,y) | -R(z,x).      10  e(x,s(y,z)) | e(x,y).
11  e(x,s(y,z)) | -e(x,z).             12  R(x,x).
13  -e($f2(x),x) | $ans(x).
14 [binary,12,8] e(x,F(y)) | -e(x,y).
15 [binary,13,11] $ans(s(x,y)) | -e($f2(s(x,y)),y).
17 [binary,13,10] $ans(s(x,y)) | e($f2(s(x,y)),x).
33 [binary,14,15] -e($f2(s(x,F(y))),y) | $ans(s(x,F(y))).
35 [binary,33,17] $ans(s(v64,F(v64))).
------------ end of proof -------------
```

In set notation, the answer is $X \subseteq F(X)$.

Example c

(c) (i) $\forall X \subseteq U : \; F(X) \subseteq X$,
(ii) R is the identity relation over a subset of U

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$\forall y \; y \in F(X) \Rightarrow y \in X$
Rewritten:	$\forall y \; (\exists x \; X(x) \wedge R(x,y))) \Rightarrow X(y)$
Negated and Clausified:	$X(\underline{x})$
	$\neg R(\underline{x},\underline{y})$
	$\neg X(\underline{y})$
X 'Resolved Away':	$\neg R(\underline{x},\underline{y})$
	$\underline{x} \neq \underline{y}$
Quantifiers Reconstructed:	$\exists x,y \; \neg R(x,y) \wedge x \neq y$
Negated again:	$\forall x,y \; R(x,y) \Rightarrow x = y$

Direction (ii) → (i)   (Bottom-Up)
Otter **Protocol**

```
set(binary_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
end_of_list.

formula_list(sos).
(all x (all y (R(x,y) -> (x = y)))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:

list(usable).
6 -e(z,F(x1)) | e($f1(z,x1),x1).    7 -e(z,F(x1)) | R($f1(z,x1),z).
8 e(z,F(x1)) | -e(x,x1) | -R(x,z).  9 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
10 e(z,s(x2,x3)) | e(z,x2).         11 e(z,s(x2,x3)) | -e(z,x3).
end_of_list.

list(sos).
13 -R(x,y) | (x = y).               14 -e($f2(x4),x4) | $ans(x4).
end_of_list.

% Heurisitc: Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).         weight(s(*1,s(*1,*1)),200).
weight(F(F(*1)),200).               weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).                 2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).        4  ($ans(s(x,x)) = $T).
5  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
6   -e(x,F(y)) | e($f1(x,y),y).     7   -e(x,F(y)) | R($f1(x,y),x).
10  e(x,s(y,z)) | e(x,y).           11  e(x,s(y,z)) | -e(x,z).
13  -R(x,y) | (x = y).              14  -e($f2(x),x) | $ans(x).
15 [binary,14,11] $ans(s(x,y)) | -e($f2(s(x,y)),y).
17 [binary,14,10] $ans(s(x,y)) | e($f2(s(x,y)),x).
21 [binary,13,7] ($f1(x,y) = x) | -e(x,F(y)).
51 [para_from,21,6] -e(x,F(y)) | e(x,y).
58 [binary,51,17] e($f2(s(F(x),y)),x) | $ans(s(F(x),y)).
59 [binary,58,15] $ans(s(F(v65),v65)).
------------ end of proof -------------
```

In set notation, the answer is $F(X) \subseteq X$.

Example d

(d) (i) $F : \mathcal{P}(U) \to \mathcal{P}(U)$ is the identity function,
(ii) $R \subseteq U^2$ is the identity relation

This example combines examples b) and c).

Example e

(e) (i) $E \subseteq U$ is a fixed point of $F : F(E) = E$,
(ii) $(\exists e \in E)[Rex]$ iff $x \in E$

Direction (i) $\rightarrow$ (ii) (Top–Down)

Logical Formulation: $\forall x\ x \in F(E) \Leftrightarrow x \in E$

Rewritten: $\forall x\ (\exists e\ e \in E \wedge R(e, x)) \Leftrightarrow x \in E$

Direction (ii) $\rightarrow$ (i) (Bottom-Up)

Otter Protocol

```
set(binary_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
end_of_list.

formula_list(sos).
(all x ((exists y (e(y,E) & R(y,x))) <-> e(x,E))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
6 -e(z,F(x1)) | e($f1(z,x1),x1).        7 -e(z,F(x1)) | R($f1(z,x1),z).
8 e(z,F(x1)) | -e(x,x1) | -R(x,z).      9 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
10 e(z,s(x2,x3)) | e(z,x2).             11 e(z,s(x2,x3)) | -e(z,x3).
end_of_list.

list(sos).
12 -e(y,E) | -R(y,x) | e(x,E).          13 e($f2(x),E) | -e(x,E).
14 R($f2(x),x) | -e(x,E).               15 -e($f3(x4),x4) | $ans(x4).
end_of_list.

% Heuristic:  Avoid double nestings of s.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).             weight(s(*1,s(*1,*1)),200).
weight(F(F(*1)),200).                   weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).                     2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).            4  ($ans(s(x,x)) = $T).
5  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
6  -e(x,F(y)) | e($f1(x,y),y).          7  -e(x,F(y)) | R($f1(x,y),x).
10  e(x,s(y,z)) | e(x,y).               11  e(x,s(y,z)) | -e(x,z).
12  -e(x,E) | -R(x,y) | e(y,E).         15  -e($f3(x),x) | $ans(x).
16 [binary,15,11] $ans(s(x,y)) | -e($f3(s(x,y)),y).
18 [binary,15,10] $ans(s(x,y)) | e($f3(s(x,y)),x).
32 [binary,18,7] $ans(s(F(x),y)) | R($f1($f3(s(F(x),y)),x),$f3(s(F(x),y))).
33 [binary,18,6] $ans(s(F(x),y)) | e($f1($f3(s(F(x),y)),x),x).
```

```
97 [binary,12,16] -e(x,E) | -R(x,$f3(s(y,E))) | $ans(s(y,E)).
223 [binary,97,33] -R($f1($f3(s(F(E),x)),E),$f3(s(y,E))) | $ans(s(y,E)) | $ans(s(F(E),x)).
224 [binary,223,32] $ans(s(F(E),E)).
------------ end of proof -------------

% We continue the search.

---------------- PROOF ----------------
8  e(x,F(y)) | -e(z,y) | -R(z,x).      10  e(x,s(y,z)) | e(x,y).
11  e(x,s(y,z)) | -e(x,z).             13  e($f2(x),E) | -e(x,E).
14  R($f2(x),x) | -e(x,E).             15  -e($f3(x),x) | $ans(x).
16 [binary,15,11] $ans(s(x,y)) | -e($f3(s(x,y)),y).
18 [binary,15,10] $ans(s(x,y)) | e($f3(s(x,y)),x).
25 [binary,16,8] $ans(s(x,F(y))) | -e(z,y) | -R(z,$f3(s(x,F(y)))).
38 [binary,13,18] e($f2($f3(s(E,x))),E) | $ans(s(E,x)).
43 [binary,14,18] R($f2($f3(s(E,x))),$f3(s(E,x))) | $ans(s(E,x)).
258 [binary,25,38] $ans(s(x,F(E))) | -R($f2($f3(s(E,y))),$f3(s(x,F(E)))) | $ans(s(E,y)).
259 [binary,258,43] $ans(s(E,F(E))).
------------ end of proof -------------
```

The result of the first proof is $F(E) \subseteq E$ and the result of the second proof is $E \subseteq F(E)$. Together we have $F(E) = E$.

Example f

(f) (i) $\forall X \subseteq U : F(F(X)) \subseteq F(X)$,
(ii) R is transitive

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$\forall z\ z \in F(F(X)) \Rightarrow z \in F(X)$
Rewritten:	$\forall z\ (\exists y\ (\exists x\ X(x) \wedge R(x,y)) \wedge R(y,z)) \Rightarrow (\exists u\ X(u) \wedge R(u,z))$
Negated and Clausified:	$X(\underline{x})$
	$R(\underline{x},\underline{y})$
	$R(\underline{y},\underline{z})$
	$\neg X(u), \neg R(u,\underline{z})$
X 'Resolved Away':	$R(\underline{x},\underline{y})$
	$R(\underline{y},\underline{z})$
	$\neg R(\underline{x},\underline{z})$
Quantifiers Reconstructed:	$\exists x,y,z\ \neg R(x,y) \wedge R(y,z) \wedge \neg R(x,z)$
Negated again:	$\forall x,y,z\ R(x,y) \wedge R(y,z) \Rightarrow R(x,z)$

Direction (ii) → (i) (Bottom-Up)

Otter **Protocol**

```
set(hyper_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
end_of_list.

formula_list(sos).
(all x (all y (all z ((R(x,y) & R(y,z)) -> R(x,z))))).
```

```
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
3 -e(z,F(x1)) | e($f1(z,x1),x1).      4 -e(z,F(x1)) | R($f1(z,x1),z).
5 e(z,F(x1)) | -e(x,x1) | -R(x,z).    6 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
7 e(z,s(x2,x3)) | e(z,x2).            8 e(z,s(x2,x3)) | -e(z,x3).
end_of_list.

list(sos).
9 -R(x,y) | -R(y,z) | R(x,z).         10 -e($f2(x4),x4) | $ans(x4).
end_of_list.

% Heuristic:  Avoid double nestings of s.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).           weight(s(*1,s(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).                   2  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
3  -e(x,F(y)) | e($f1(x,y),y).        4  -e(x,F(y)) | R($f1(x,y),x).
5  e(x,F(y)) | -e(z,y) | -R(z,x).     7  e(x,s(y,z)) | e(x,y).
8  e(x,s(y,z)) | -e(x,z).             9  -R(x,y) | -R(y,z) | R(x,z).
10  -e($f2(x),x) | $ans(x).
12 [hyper,10,7] $ans(s(x,y)) | e($f2(s(x,y)),x).
15 [hyper,12,4] $ans(s(F(x),y)) | R($f1($f2(s(F(x),y)),x),$f2(s(F(x),y))).
16 [hyper,12,3] $ans(s(F(x),y)) | e($f1($f2(s(F(x),y)),x),x).
19 [hyper,16,4] $ans(s(F(F(x)),y))
             R($f1($f1($f2(s(F(F(x)),y)),F(x)),x),$f1($f2(s(F(F(x)),y)),F(x))).
20 [hyper,16,3] $ans(s(F(F(x)),y)) | e($f1($f1($f2(s(F(F(x)),y)),F(x)),x),x).
115 [hyper,19,9,15] $ans(s(F(F(x)),y)) |
             R($f1($f1($f2(s(F(F(x)),y)),F(x)),x),$f2(s(F(F(x)),y))).
171 [hyper,115,5,20] $ans(s(F(F(x)),y)) | e($f2(s(F(F(x)),y)),F(x)).
174 [hyper,171,8] $ans(s(F(F(x)),y)) | e($f2(s(F(F(x)),y)),s(z,F(x))).
175 [binary,174,10] $ans(s(F(F(x)),F(x))).
------------ end of proof -------------
```

In set notation, the answer is $F(F(X)) \subseteq F(X)$.

Example g

(g) (i) $\forall X \subseteq U: \ F(X) \subseteq F(F(X))$,
 (ii) R is dense $[\forall x, y \in U : Rxy \Rightarrow (\exists z)[Rxz \wedge Rzy]]$

Direction (i) $\rightarrow$ (ii) (Top–Down)

Logical Formulation:	$\forall y \ y \in F(X) \Rightarrow y \in F(F(X))$
Rewritten:	$\forall y \ (\exists x \ X(x) \wedge R(x,y)) \Rightarrow (\exists z \ (\exists u \ X(u) \wedge R(u,z)) \wedge R(z,y))$
Negated and Clausified:	$X(\underline{x})$
	$R(\underline{x}, \underline{y})$
	$\neg X(u), \neg R(u,z), \neg R(z,\underline{y})$
X ‘Resolved Away’:	$R(\underline{x}, \underline{y})$

$\neg R(\underline{x}, z), \neg R(z, \underline{y})$

Quantifiers Reconstructed: $\exists x, y\ R(x, y) \wedge (\forall z\ \neg R(x, z) \vee R(z, y))$

Negated again: $\forall x, y\ R(x, y) \Rightarrow (\exists z\ R(x, z) \wedge R(z, y))$

Direction (ii) → (i) (Bottom-Up)

Otter Protocol

```
set(hyper_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
end_of_list.

formula_list(sos).
(all x (all y (R(x,y) -> (exists z (R(x,z) & R(z,y)))))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
7 -e(z,F(x1)) | e($f1(z,x1),x1).     8 -e(z,F(x1)) | R($f1(z,x1),z).
9 e(z,F(x1)) | -e(x,x1) | -R(x,z).   10 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
11 e(z,s(x2,x3)) | e(z,x2).          12 e(z,s(x2,x3)) | -e(z,x3).
end_of_list.

list(sos).
13 -R(x,y) | R(x,$f2(x,y)).          14 -R(x,y) | R($f2(x,y),y).
15 -e($f3(x4),x4) | $ans(x4).
end_of_list.

% Heuristic:  Avoid double nestings of s.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).          weight(s(*1,s(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).                2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).       4  ($ans(s(x,x)) = $T).
5  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
6  -e(x,F(y)) | e($f1(x,y),y).     7  -e(x,F(y)) | R($f1(x,y),x).
8  e(x,F(y)) | -e(z,y) | -R(z,x).  10  e(x,s(y,z)) | e(x,y).
11  e(x,s(y,z)) | -e(x,z).         12  -R(x,y) | R(x,$f2(x,y)).
13  -R(x,y) | R($f2(x,y),y).       14  -e($f3(x),x) | $ans(x).
16 [hyper,14,10] $ans(s(x,y)) | e($f3(s(x,y)),x).
19 [hyper,16,7] $ans(s(F(x),y)) | R($f1($f3(s(F(x),y)),x),$f3(s(F(x),y))).
20 [hyper,16,6] $ans(s(F(x),y)) | e($f1($f3(s(F(x),y)),x),x).
25 [hyper,19,13] $ans(s(F(x),y)) |
           R($f2($f1($f3(s(F(x),y)),x),$f3(s(F(x),y))),$f3(s(F(x),y))).
26 [hyper,19,12] $ans(s(F(x),y)) |
           R($f1($f3(s(F(x),y)),x),$f2($f1($f3(s(F(x),y)),x),$f3(s(F(x),y)))).
219 [hyper,26,8,20] $ans(s(F(x),y)) | e($f2($f1($f3(s(F(x),y)),x),$f3(s(F(x),y))),F(x)).
```

```
223 [hyper,219,8,25] $ans(s(F(x),y)) | e($f3(s(F(x),y)),F(F(x))).
226 [hyper,223,11] $ans(s(F(x),y)) | e($f3(s(F(x),y)),s(z,F(F(x)))).
227 [binary,226,14] $ans(s(F(x),F(F(x)))).
------------ end of proof -------------
```

In set notation, the answer is $F(X) \subseteq F(F(X))$.

Example h

(h) (i) $\forall X, Y \subseteq U:\ F(X) \bigcap Y = \emptyset$ iff $X \bigcap F(Y) = \emptyset$,
(ii) R is symmetric

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$(\neg(\exists x\ x \in F(X) \wedge x \in Y)) \Rightarrow (\neg(\exists x\ x \in X \wedge x \in F(Y)))$
Rewritten:	$(\neg(\exists x\ (\exists z\ z \in X \wedge R(z,x)) \wedge x \in Y))$
	$\Rightarrow (\neg(\exists x\ x \in X \wedge (\exists y\ y \in Y \wedge R(y,x))))$
Negated and Clausified:	$\neg X(z), \neg R(z,x), Y(x)$
	$X(\underline{x})$
	$Y(\underline{y})$
	$R(\underline{y}, \underline{x})$
X, Y 'Resolved Away':	$\neg R(\underline{x}, \underline{y})$
	$R(\underline{y}, \underline{x})$
Quantifiers Reconstructed:	$\exists x, y\ \neg R(x,y) \wedge R(y,x)$
Negated again:	$\forall x, y\ R(x,y) \Rightarrow R(y,x)$

Direction (ii) → (i) (Bottom-Up)
An alternative formulation for (i) is $F(F(X')') \subseteq X$ where X' means the complement of X. By axiomatizing the complement function (c) we aim for this version.

Otter **Protocol**

```
set(binary_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x (all X (e(x,c(X)) <-> -e(x,X)))).
end_of_list.

formula_list(sos).
(all x (all y (R(x,y) -> R(y,x)))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
9 -e(z,F(x1)) | e($f1(z,x1),x1).      10 -e(z,F(x1)) | R($f1(z,x1),z).
11 e(z,F(x1)) | -e(x,x1) | -R(x,z).   12 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
13 e(z,s(x2,x3)) | e(z,x2).           14 e(z,s(x2,x3)) | -e(z,x3).
15 -e(x,c(x4)) | -e(x,x4).            16 e(x,c(x4)) | e(x,x4).
end_of_list.

list(sos).
17 -R(x,y) | R(y,x).                  18 -e($f2(x5),x5) | $ans(x5).
```

```
end_of_list.

% Heuristic:  Avoid double nestings of s,F and c.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).          weight(s(*1,s(*1,*1)),200).
weight(F(F(*1)),200).                weight(c(c(*1)),200).
weight(c(s(*1,*1)),200).             weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).                  2  (c(c(x)) = junk).
3  ($ans(s(s(x,y),z)) = $T).         4  ($ans(s(x,s(y,z))) = $T).
5  ($ans(s(x,x)) = $T).              6  ($ans(F(x)) = $T).
7  ($ans(c(x)) = $T).                8  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
9  -e(x,F(y)) | e($f1(x,y),y).     10  -e(x,F(y)) | R($f1(x,y),x).
11  e(x,F(y)) | -e(z,y) | -R(z,x). 13  e(x,s(y,z)) | e(x,y).
14  e(x,s(y,z)) | -e(x,z).         15  -e(x,c(y)) | -e(x,y).
16  e(x,c(y)) | e(x,y).            17  -R(x,y) | R(y,x).
18  -e($f2(x),x) | $ans(x).
20 [binary,18,14] $ans(s(x,y)) | -e($f2(s(x,y)),y).
22 [binary,18,13] $ans(s(x,y)) | e($f2(s(x,y)),x).
29 [binary,17,11] -R(x,y) | e(x,F(z)) | -e(y,z).
30 [binary,20,16] $ans(s(x,y)) | e($f2(s(x,y)),c(y)).
49 [binary,22,10] $ans(s(F(x),y)) | R($f1($f2(s(F(x),y)),x),$f2(s(F(x),y))).
50 [binary,22,9] $ans(s(F(x),y)) | e($f1($f2(s(F(x),y)),x),x).
92 [binary,50,15] $ans(s(F(c(x)),y)) | -e($f1($f2(s(F(c(x)),y)),c(x)),x).
137 [binary,29,30] -R(x,$f2(s(y,z))) | e(x,F(c(z))) | $ans(s(y,z)).
802 [binary,137,92] -R($f1($f2(s(F(c(F(c(x)))),y)),c(F(c(x)))),$f2(s(z,x))) |
                              $ans(s(z,x)) | $ans(s(F(c(F(c(x)))),y)).
803 [binary,802,49] $ans(s(F(c(F(c(v65)))),v65)).
------------ end of proof -------------
```

In set notation, the answer is $F(F(X')') \subseteq X$.

Example i

(i) (i) F maps singletons onto singletons,
(ii) R is a unary operation over U

Direction (i) $\rightarrow$ (ii) (Top–Down)

Logical Formulation: $(\forall x, y\ x \in X \wedge y \in X \Rightarrow x = y)$
$\Rightarrow (\forall y, z\ y \in F(X) \wedge z \in F(X) \Rightarrow y = z)$

Rewritten: $(\forall x, y\ x \in X \wedge y \in X \Rightarrow x = y)$
$\Rightarrow (\forall y, z\ (\exists x\ x \in X \wedge R(x, y)) \wedge (\exists x'\ x' \in X \wedge R(x', z)) \Rightarrow y = z)$

Negated and Clausified: $\neg X(x), \neg X(y), x = y$
$X(\underline{x})$
$R(\underline{x}, \underline{y})$
$X(\underline{x'})$
$R(\underline{x'}, \underline{z})$
$\underline{y} \neq \underline{z}$

X 'Resolved Away': $\neg R(\underline{x}, \underline{y})$
$R(\underline{x'}, \underline{z})$

$\underline{y} \neq \underline{z}$
$\underline{x} \neq \underline{x'}$
Quantifiers Reconstructed: $\exists x, x', y, z\ R(x,y) \wedge R(x,z) \wedge y \neq z \wedge x \neq x'$
Negated again: $\forall x, x', y, z\ R(x,y) \wedge R(x,z) \wedge x = x' \Rightarrow y = z$ which is equivalent to $\forall x, y, z\ R(x,y) \wedge R(x,z) \wedge \Rightarrow y = z$

Direction (ii) → (i) (Bottom-Up)

OTTER Protocol

```
set(binary_res).

formula_list(usable).
(all w (all z (all X (e(w,z,F(X)) <-> (exists x (e(w,x,X) & R(x,z))))))).
(all w (all X (all Y (H(w,i(X,Y)) <-> (H(w,X) -> H(w,Y)))))).
(all x (x = x)).
(all w (all X (H(w,sg(X)) <-> (all x (all y ((e(w,x,X) & e(w,y,X)) -> (x = y))))))).
end_of_list.

formula_list(sos).
(all x (all y (all z ((R(x,y) & R(x,z)) -> (y = z))))).
-(exists f (all w (H(w,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
7 -e(w,z,F(x1)) | e(w,$f1(w,z,x1),x1).  8 -e(w,z,F(x1)) | R($f1(w,z,x1),z).
9 e(w,z,F(x1)) | -e(w,x,x1) | -R(x,z).  10-H(w,i(x2,x3)) | -H(w,x2) | H(w,x3).
11 H(w,i(x2,x3)) | H(w,x2).             12 H(w,i(x2,x3)) | -H(w,x3).
13 (x = x).
14 -H(w,sg(x4)) | -e(w,x,x4) | -e(w,y,x4) | (x = y).
15 H(w,sg(x4)) | e(w,$f3(w,x4),x4).      16 H(w,sg(x4)) | e(w,$f2(w,x4),x4).
17 H(w,sg(x4)) | ($f3(w,x4) != $f2(w,x4)).
end_of_list.

list(sos).
18 -R(x,y) | -R(x,z) | (y = z).         19 -H($f4(x5),x5) | $ans(x5).
end_of_list.

% Heuristic:  Avoid double nestings of i and F.
weight_list(purge_gen).
weight(i(i(*1,*1),*1),200).             weight(i(*1,i(*1,*1)),200).
weight(F(F(*1)),200).                   weight(sg(s(*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (i(x,x) = junk).                     2  ($ans(i(i(x,y),z)) = $T).
3  ($ans(i(x,i(y,z))) = $T).            4  ($ans(i(x,x)) = $T).
5  ($ans(sg(x)) = $T).                   6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
7  -e(x,y,F(z)) | e(x,$f1(x,y,z),z).    8  -e(x,y,F(z)) | R($f1(x,y,z),y).
10  -H(x,i(y,z)) | -H(x,y) | H(x,z).    11  H(x,i(y,z)) | H(x,y).
12  H(x,i(y,z)) | -H(x,z).
14  -H(x,sg(y)) | -e(x,z,y) | -e(x,u,y) | (z = u).
```

```
15  H(x,sg(y)) | e(x,$f3(x,y),y).        16  H(x,sg(y)) | e(x,$f2(x,y),y).
17  H(x,sg(y)) | ($f3(x,y) != $f2(x,y)). 18  -R(x,y) | -R(x,z) | (y = z).
19  -H($f4(x),x) | $ans(x).
20 [binary,19,12] $ans(i(x,y)) | -H($f4(i(x,y)),y).
22 [binary,19,11] $ans(i(x,y)) | H($f4(i(x,y)),x).
26 [binary,20,16] $ans(i(x,sg(y))) | e($f4(i(x,sg(y))),$f2($f4(i(x,sg(y))),y),y).
27 [binary,20,15] $ans(i(x,sg(y))) | e($f4(i(x,sg(y))),$f3($f4(i(x,sg(y))),y),y).
28 [binary,20,11] $ans(i(x,y)) | H($f4(i(x,y)),i(y,z)).
31 [binary,22,14] $ans(i(sg(x),y)) | -e($f4(i(sg(x),y)),z,x) |
       -e($f4(i(sg(x),y)),u,x) | (z = u).
37 [binary,18,8] -R($f1(x,y,z),u) | (u = y) | -e(x,y,F(z)).
46 [binary,26,7] $ans(i(x,sg(F(y)))) |
           e($f4(i(x,sg(F(y)))),$f1($f4(i(x,sg(F(y)))), $f2($f4(i(x,sg(F(y)))),F(y)),y),y).
48 [binary,27,8] $ans(i(x,sg(F(y)))) |
          R($f1($f4(i(x,sg(F(y)))),$f3($f4(i(x,sg(F(y)))),F(y)),y),
            $f3($f4(i(x,sg(F(y)))),F(y))).
66 [binary,37,16] -R($f1(x,$f2(x,F(y)),y),z) | (z = $f2(x,F(y))) | H(x,sg(F(y))).
115 [binary,31,7] $ans(i(sg(x),y)) | -e($f4(i(sg(x),y)),z,x) | ($f1($f4(i(sg(x),y)),u,x) = z) |
       -e($f4(i(sg(x),y)),u,F(x)).
203 [binary,66,17] -R($f1(x,$f2(x,F(y)),y),$f3(x,F(y))) | H(x,sg(F(y))).
212 [binary,203,10] -R($f1(x,$f2(x,F(y)),y),$f3(x,F(y))) | -H(x,i(sg(F(y)),z)) | H(x,z).
452 [para_from,115,48,unit_del,27] $ans(i(sg(x),sg(F(x)))) |
        R(y,$f3($f4(i(sg(x),sg(F(x)))),F(x))) |
       -e($f4(i(sg(x),sg(F(x)))),y,x) | $ans(i(sg(x),sg(F(x)))).
499 [binary,452,212,unit_del,46,28] $ans(i(sg(x),sg(F(x)))) | H($f4(i(sg(x),sg(F(x)))),y) |
         $ans(i(sg(x),sg(F(x)))) | $ans(i(sg(x),sg(F(x)))).
500 [binary,499,20] $ans(i(sg(x),sg(F(x)))).
------------ end of proof -------------
```

In set notation the answer is $singleton(X)$ implies $singleton(F(X))$.

Example j

(j) (i) F is an involution over $\mathcal{P}(U)$, i.e. $F(F(X)) = X$
 (ii) R is an involution over U, i.e.
 $\forall x, y, z\ R(x, y) \wedge R(y, z) \Rightarrow x = z$ and $\forall x\ \exists y\ R(x, y) \wedge R(y, x)$.

We split the equation into the two parts $F(F(X)) \subseteq X$ and $X \subseteq F(F(X))$.

First Part: $F(F(X)) \subseteq X$

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$\forall z\ z \in F(F(X)) \Rightarrow z \in X$
Rewritten:	$\forall z\ (\exists y\ (\exists x\ x \in X \wedge R(x, y)) \wedge R(y, z)) \Rightarrow z \in X$
Negated and Clausified:	$X(\underline{x})$
	$R(\underline{x}, \underline{y})$
	$R(\underline{y}, \underline{z})$
	$\neg X(\underline{z})$
X 'Resolved Away':	$R(\underline{x}, \underline{y})$
	$R(\underline{y}, \underline{z})$
	$\underline{x} \neq \underline{z}$
Quantifiers Reconstructed:	$\exists x, y, z\ R(x, y) \wedge R(y, z) \wedge x \neq z$
Negated again:	$\forall x, y, z\ R(x, y) \wedge R(y, z) \Rightarrow x = z$

Direction (ii) → (i) (Bottom-Up)

Otter Protocol

```
set(binary_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x (x = x)).
end_of_list.

formula_list(sos).
(all x (all y (all z ((R(x,y) & R(y,z)) -> (x = z))))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
7 -e(z,F(x1)) | e($f1(z,x1),x1).    8 -e(z,F(x1)) | R($f1(z,x1),z).
9 e(z,F(x1)) | -e(x,x1) | -R(x,z). 10 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
11 e(z,s(x2,x3)) | e(z,x2).       12 e(z,s(x2,x3)) | -e(z,x3).
13 (x = x).
end_of_list.

list(sos).
14 -R(x,y) | -R(y,z) | (x = z).    15 -e($f2(x4),x4) | $ans(x4).
end_of_list.

% Heuristic:  Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).       weight(s(*1,s(*1,*1)),200).
weight(F(F(*1)),200).             weight(F(s(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).               2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).      4  ($ans(s(x,x)) = $T).
5  ($ans(F(x)) = $T).             6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
7  -e(x,F(y)) | e($f1(x,y),y).     8  -e(x,F(y)) | R($f1(x,y),x).
11  e(x,s(y,z)) | e(x,y).         12  e(x,s(y,z)) | -e(x,z).
14  -R(x,y) | -R(y,z) | (x = z).  15  -e($f2(x),x) | $ans(x).
16 [binary,15,12] $ans(s(x,y)) | -e($f2(s(x,y)),y).
18 [binary,15,11] $ans(s(x,y)) | e($f2(s(x,y)),x).
30 [binary,18,8] $ans(s(F(x),y)) | R($f1($f2(s(F(x),y)),x),$f2(s(F(x),y))).
31 [binary,18,7] $ans(s(F(x),y)) | e($f1($f2(s(F(x),y)),x),x).
32 [binary,14,8] -R(x,y) | ($f1(x,z) = y) | -e(x,F(z)).
75 [para_from,32,7] -e(x,F(y)) | e(z,y) | -R(x,z).
107 [binary,75,16] -e(x,F(y)) | -R(x,$f2(s(z,y))) | $ans(s(z,y)).
533 [binary,30,107] $ans(s(F(x),y)) | -e($f1($f2(s(F(x),y)),x),F(y)).
534 [binary,533,31] $ans(s(F(F(v65)),v65)).
------------ end of proof -------------
```

In set notation, the answer is $F(F(X)) \subseteq X$.

Second Part: $X \subseteq F(F(X))$

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$\forall x\ x \in X \Rightarrow x \in F(F(X))$
Rewritten:	$\forall x\ x \in X \Rightarrow (\exists y\ (\exists z\ z \in X \wedge R(z,x)) \wedge R(y,x))$
Negated and Clausified:	$X(\underline{x})$
	$X(z), \neg R(z,y), \neg R(y,\underline{x})$
X 'Resolved Away':	$\neg R(\underline{x},y), \neg R(y,\underline{x})$
Quantifiers Reconstructed:	$\exists x\ \forall y\ \neg R(x,y) \vee \neg R(y,x)$
Negated again:	$\forall x\ \exists y\ R(x,y) \wedge R(y,x)$

Direction (ii) → (i) (Bottom-Up)

OTTER **Protocol**

```
set(binary_res).

formula_list(usable).
(all z (all X (e(z,F(X)) <-> (exists x (e(x,X) & R(x,z)))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x (x = x)).
end_of_list.

formula_list(sos).
(all x (exists y (R(x,y) & R(y,x)))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
7 -e(z,F(x1)) | e($f1(z,x1),x1).    8 -e(z,F(x1)) | R($f1(z,x1),z).
9 e(z,F(x1)) | -e(x,x1) | -R(x,z). 10 -e(z,s(x2,x3)) | -e(z,x2) | e(z,x3).
11 e(z,s(x2,x3)) | e(z,x2).        12 e(z,s(x2,x3)) | -e(z,x3).
13 (x = x).
end_of_list.

list(sos).
14 R(x,$f2(x)).                    15 R($f2(x),x).
16 -e($f3(x4),x4) | $ans(x4).
end_of_list.

% Heuristic:  Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).        weight(s(*1,s(*1,*1)),200).
weight(F(F(*1)),200).              weight(F(s(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).                2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).       4  ($ans(s(x,x)) = $T).
5  ($ans(F(x)) = $T).              6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
9  e(x,F(y)) | -e(z,y) | -R(z,x).  11  e(x,s(y,z)) | e(x,y).
```

```
12 e(x,s(y,z)) | -e(x,z).          14  R(x,$f2(x)).
15 R($f2(x),x).                    16  -e($f3(x),x) | $ans(x).
17 [binary,14,9] e($f2(x),F(y)) | -e(x,y).
18 [binary,15,9] e(x,F(y)) | -e($f2(x),y).
19 [binary,16,12] $ans(s(x,y)) | -e($f3(s(x,y)),y).
21 [binary,16,11] $ans(s(x,y)) | e($f3(s(x,y)),x).
39 [binary,17,21] e($f2($f3(s(x,y))),F(x)) | $ans(s(x,y)).
43 [binary,18,19] -e($f2($f3(s(x,F(y)))),y) | $ans(s(x,F(y))).
44 [binary,43,39] $ans(s(v64,F(F(v64)))).
------------ end of proof -------------
```

In set notation, the answer is $X \subseteq F(F(X))$.

8.2 Ternary Relations

We continue with the examples of lemma 2.6. They correlate a ternary relation R with a binary function F. The definition of F for this case (def. 2.1) is used as a rewrite rule:

(r1) $\quad (z \in F(X,Y)) \to (\exists x, y\ x \in X \wedge y \in Y \wedge R(x,y,z))$

Example a

(a) (i) F has no divisors of zero $[\forall X, Y \subseteq U:\ F(X,Y) = \emptyset \Rightarrow X = \emptyset$ or $Y = \emptyset]$,
(ii) $\forall x, y\ \in U, \exists z \in U$ such that $Rxyz$

Direction (i) $\to$ (ii) (Top–Down)

Logical Formulation:	$(\neg\exists z\ z \in F(X,Y)) \Rightarrow ((\neg\exists x\ x \in X) \vee (\neg\exists y\ y \in Y))$
Rewritten:	$(\neg\exists z\ (\exists x, y\ x \in X \wedge y \in Y \wedge R(x,y,z))) \Rightarrow ((\neg\exists x\ x \in X) \vee (\neg\exists y\ y \in Y))$
Negated and Clausified:	$\neg X(x), \neg Y(y), \neg R(x,y,z)$
	$X(\underline{x})$
	$Y(\underline{y})$
X, Y 'Resolved Away':	$\neg R(\underline{x}, \underline{y}, z)$
Quantifiers Reconstructed:	$\exists x, y\ \forall z\ \neg R(x,y,z)$
Negated again:	$\forall x, y\ \exists z\ R(x,y,z))$

Direction (ii) $\to$ (i) (Bottom-Up)

First trial: Quantifier Elimination

$\exists R\ \Gamma(R) \wedge Def(F,R) =$

$\exists R\ (\forall x, y\ \exists z\ R(x,y,z)) \wedge (\forall z\ e(z, F(X,Y)) \Leftarrow \exists x, y\ e(x,X) \wedge e(y,Y) \wedge R(x,y,z))$

(the '$\Leftarrow$' part of the definition of F is irrelevant.)

Clauses:

$R(x,y,f(x,y))$

$\neg e(x,X), \neg e(y,Y), \neg R(x,y,z), e(z, F(X,Y))$

R resolved away

$\neg e(x,X), \neg e(y,Y), e(f(x,y), F(X,Y))$

Quantifiers reconstructed:

$\forall x, y\ e(x,X) \wedge e(y,Y) \Rightarrow \exists z\ e(z, F(X,Y))$

The set notation of this formula is:

$(\exists x\ x \in X \wedge \exists y\ y \in Y \Rightarrow (\exists z\ z \in F(X,Y))$

whose contraposition means nothing else than

$F(X,Y) \neq \emptyset \Rightarrow F(X) \neq \emptyset \vee F(Y) \neq \emptyset$

Although the quantifier elimination trial was successful, we also show again the theorem prover version. It illustrates the use of the world dependent *Holds*–predicate in a more complex setting.

OTTER **Protocol:**

```
set(binary_res).

formula_list(usable).
(all w all z all X all Y (e(w,z,F(X,Y)) <->
          (exists x exists y (e(w,x,X) & e(w,y,Y) & R(x,y,z))))).
(all w (all X (all Y (H(w,i(X,Y)) <-> (H(w,X) -> H(w,Y)))))).
(all w (all X (all Y (H(w,or(X,Y)) <-> (H(w,X) | H(w,Y)))))).
(all w (all X (H(w,ep(X)) <-> (all x -e(w,x,X))))).
end_of_list.

formula_list(sos).
(all a (all b (exists c R(a,b,c)))).
-(exists f (all w (H(w,f) & -$ans(f)))).
end_of_list.

Clauses:
list(usable).
11 -e(w,z,F(x1,x2)) | e(w,$f2(w,z,x1,x2),x1).
12 -e(w,z,F(x1,x2)) | e(w,$f1(w,z,x1,x2),x2).
13 -e(w,z,F(x1,x2)) | R($f2(w,z,x1,x2),$f1(w,z,x1,x2),z).
14 e(w,z,F(x1,x2)) | -e(w,x,x1) | -e(w,y,x2) | -R(x,y,z).
15 -H(w,i(x3,x4)) | -H(w,x3) | H(w,x4).
16 H(w,i(x3,x4)) | H(w,x3).
17 H(w,i(x3,x4)) | -H(w,x4).
18 -H(w,or(x5,x6)) | H(w,x5) | H(w,x6).
19 H(w,or(x5,x6)) | -H(w,x5).
20 H(w,or(x5,x6)) | -H(w,x6).
21 -H(w,ep(x7)) | -e(w,x,x7).
22 H(w,ep(x7)) | e(w,$f3(w,x7),x7).
end_of_list.

list(sos).
23 R(x8,x9,$f4(x8,x9)).
24 -H($f5(x10),x10) | $ans(x10).
end_of_list.

% Heuristic:  Avoid double nestings of i, or, ep and F
weight_list(purge_gen).
weight(i(i(*1,*1),*1),200).        weight(i(*1,i(*1,*1)),200).
weight(i(or(*1,*1),*1),200).       weight(or(*1,ep(F(*1,*1))),200).
weight(or(ep(F(*1,*1)),*1),200).   weight(or(or(*1,*1),*1),200).
weight(or(*1,or(*1,*1)),200).      weight(or(i(*1,*1),*1),200).
weight(or(*1,i(*1,*1)),200).       weight(ep(i(*1,*1)),200).
weight(ep(or(*1,*1)),200).         weight(F(i(*1,*1),*1),200).
weight(F(*1,i(*1,*1)),200).        weight(F(or(*1,*1),*1),200).
weight(F(*1,or(*1,*1)),200).       weight(F(ep(*1),*1),200).
weight(F(*1,ep(*1)),200).          weight(F(F(*1,*1),*1),200).
weight(F(*1,F(*1,*1)),200).        weight(ep(ep(*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
```

```
1  (i(x,x) = junk).             2  (i(x,or(x,y)) = junk).
3  (F(x,x) = junk).             4  ($ans(i(i(x,y),z)) = $T).
5  ($ans(i(x,i(y,z))) = $T).    6  ($ans(i(x,x)) = $T).
7  ($ans(or(x,y)) = $T).        8  ($ans(ep(x)) = $T).
9  ($ans(F(x,y)) = $T).        10  ($ans(junk) = $T).
end_of_list.

----> UNIT CONFLICT at 6699.19 sec
---------------- PROOF ----------------
14  e(x,y,F(z,u)) | -e(x,v,z) | -e(x,w,u) | -R(v,w,y).
16  H(x,i(y,z)) | H(x,y).       17  H(x,i(y,z)) | -H(x,z).
19  H(x,or(y,z)) | -H(x,y).     20  H(x,or(y,z)) | -H(x,z).
21  -H(x,ep(y)) | -e(x,z,y).    22  H(x,ep(y)) | e(x,$f3(x,y),y).
23  R(x,y,$f4(x,y)).            24  -H($f5(x),x) | $ans(x).
27 [binary,24,17] $ans(i(x,y)) | -H($f5(i(x,y)),y).
29 [binary,24,16] $ans(i(x,y)) | H($f5(i(x,y)),x).
32 [binary,23,14] e(x,$f4(y,z),F(u,v)) | -e(x,y,u) | -e(x,z,v).
34 [binary,27,20] $ans(i(x,or(y,z))) | -H($f5(i(x,or(y,z))),z).
36 [binary,27,19] $ans(i(x,or(y,z))) | -H($f5(i(x,or(y,z))),y).
43 [binary,29,21] $ans(i(ep(x),y)) | -e($f5(i(ep(x),y)),z,x).
60 [binary,34,22] $ans(i(x,or(y,ep(z)))) |
                  e($f5(i(x,or(y,ep(z)))),$f3($f5(i(x,or(y,ep(z)))),z),z).
68 [binary,36,22] $ans(i(x,or(ep(y),z))) |
                  e($f5(i(x,or(ep(y),z))),$f3($f5(i(x,or(ep(y),z))),y),y).
750 [binary,32,43] -e($f5(i(ep(F(x,y)),z)),u,x) | -e($f5(i(ep(F(x,y)),z)),v,y) |
         $ans(i(ep(F(x,y)),z)).
14468 [binary,750,68] -e($f5(i(ep(F(x,y)),or(ep(x),z))),u,y) | $ans(i(ep(F(x,y)),or(ep(x),z))).
14473 [binary,14468,60] $ans(i(ep(F(x,v66)),or(ep(x),ep(v66)))).
------------ end of proof -------------
```

In set notation, the answer is $F(X,Y) = \emptyset \Rightarrow X = \emptyset \vee Y = \emptyset$.

Example b

(b) (i) F is commutative $[\forall X, Y \subseteq U :\ F(X,Y) = F(Y,X)]$,
    (ii) F is $(1,2)$-symmetric $[\forall x, y, z\ \subseteq U :\ Rxyz \Rightarrow Ryxz]$

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$\forall z\ z \in F(X,Y) \Rightarrow z \in F(Y,X)$
Rewritten:	$\forall z\ (\exists x, y\ x \in X \wedge y \in Y \wedge R(x,y,z)) \Rightarrow (\exists y, x\ x \in X \wedge y \in Y \wedge R(y,x,z))$
Negated and Clausified:	$X(\underline{x})$
	$Y(\underline{y})$
	$R(\underline{x}, \underline{y}, \underline{z})$
	$\neg X(x), \neg Y(y), \neg R(y,x,\underline{z})$
X, Y 'Resolved Away':	$R(\underline{x}, \underline{y}, \underline{z})$
	$\neg R(\underline{y}, \underline{x}, \underline{z})$
Quantifiers Reconstructed:	$\exists x, y, z\ R(x,y,z) \wedge \neg R(y,x,z)$
Negated again:	$\forall x, y, z\ R(x,y,z) \Rightarrow R(y,x,z))$

Direction (ii) → (i) (Bottom-Up)

Otter **Protocol:**

```
set(ur_res).

formula_list(sos).
```

```
(all w all z all X all Y (e(w,z,F(X,Y)) <->
          (exists x exists y (e(w,x,X) & e(w,y,Y) & R(x,y,z))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x all y all z (R(x,y,z) -> R(y,x,z))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(sos).
7 -e(z,F(x1,x2)) | e($f2(z,x1,x2),x1).
8 -e(z,F(x1,x2)) | e($f1(z,x1,x2),x2).
9 -e(z,F(x1,x2)) | R($f2(z,x1,x2),$f1(z,x1,x2),z).
10 e(z,F(x1,x2)) | -e(x,x1) | -e(y,x2) | -R(x,y,z).
11 -e(z,s(x3,x4)) | -e(z,x3) | e(z,x4).
12 e(z,s(x3,x4)) | e(z,x3).
13 e(z,s(x3,x4)) | -e(z,x4).
14 -R(x5,x6,x7) | R(x6,x5,x7).
15 -e($f3(x8),x8) | $ans(x8).
end_of_list.

% Heuristic:  Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).      weight(s(*1,s(*1,*1)),200).
weight(F(s(*1,*1),*1),200).      weight(F(*1,s(*1,*1)),200).
weight(F(F(*1,*1),*1),200).      weight(F(*1,F(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).          2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T). 4  ($ans(s(x,x)) = $T).
5  ($ans(F(x,y)) = $T).      6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
7  -e(x,F(y,z)) | e($f2(x,y,z),y).
8  -e(x,F(y,z)) | e($f1(x,y,z),z).
9  -e(x,F(y,z)) | R($f2(x,y,z),$f1(x,y,z),x).
10  e(x,F(y,z)) | -e(u,y) | -e(v,z) | -R(u,v,x).
12  e(x,s(y,z)) | e(x,y).
13  e(x,s(y,z)) | -e(x,z).
14  -R(x,y,z) | R(y,x,z).
15  -e($f3(x),x) | $ans(x).
17 [ur,12,15] e($f3(s(x,y)),x) | $ans(s(x,y)).
20 [ur,13,15] -e($f3(s(x,y)),y) | $ans(s(x,y)).
25 [ur,7,17] e($f2($f3(s(F(x,y),z)),x,y),x) | $ans(s(F(x,y),z)).
26 [ur,8,17] e($f1($f3(s(F(x,y),z)),x,y),y) | $ans(s(F(x,y),z)).
29 [ur,9,17] R($f2($f3(s(F(x,y),z)),x,y),$f1($f3(s(F(x,y),z)),x,y),$f3(s(F(x,y),z))) |
      $ans(s(F(x,y),z)).
31 [ur,10,20,26,25] -R($f1($f3(s(F(x,y),z)),x,y),$f2($f3(s(F(u,v),w)),u,v),$f3(s(v6,F(y,u)))) |
      $ans(s(v6,F(y,u))) | $ans(s(F(x,y),z)) | $ans(s(F(u,v),w)).
45 [ur,29,14] $ans(s(F(x,y),z)) |
      R($f1($f3(s(F(x,y),z)),x,y),$f2($f3(s(F(x,y),z)),x,y),$f3(s(F(x,y),z))).
46 [binary,45,31] $ans(s(F(v67,v68),F(v68,v67))).
------------ end of proof -------------
```

In set notation, the answer is $F(X,Y) \subseteq F(Y,X)$. Since this is symmetric, we can conclude $F(X,Y) = F(Y,X)$.

Example c

(c) (i) F is upper semi-idempotent $[\forall X \subseteq U: \ X \subseteq F(X,X)]$,
(ii) R is totally reflexive $[\forall x \subseteq U: \ Rxxx]$

Direction (i) $\rightarrow$ (ii) (Top–Down)

Logical Formulation:	$(\exists x \; x \in X \Rightarrow x \in F(X,X)$
Rewritten:	$(\exists x \; x \in X \Rightarrow \neg(\exists x, y \; y \in X \wedge y \in X \wedge R(x,y,x))$
Negated and Clausified:	$X(\underline{x})$
	$\neg X(x), \neg X(y), \neg R(x,y,\underline{x})$
X 'Resolved Away':	$\neg R(\underline{x},\underline{x},\underline{x})$
Quantifiers Reconstructed:	$\exists x \; \neg R(x,x,x)$
Negated again:	$\forall x \; R(x,x,x))$

Direction (ii) $\rightarrow$ (i) (Bottom-Up)

Otter **Protocol:**

```
set(ur_res).

formula_list(sos).
(all w all z all X all Y (e(w,z,F(X,Y)) <->
          (exists x exists y (e(w,x,X) & e(w,y,Y) & R(x,y,z))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x R(x,x,x)).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(sos).
7 -e(z,F(x1,x2)) | e($f2(z,x1,x2),x1).
8 -e(z,F(x1,x2)) | e($f1(z,x1,x2),x2).
9 -e(z,F(x1,x2)) | R($f2(z,x1,x2),$f1(z,x1,x2),z).
10 e(z,F(x1,x2)) | -e(x,x1) | -e(y,x2) | -R(x,y,z).
11 -e(z,s(x3,x4)) | -e(z,x3) | e(z,x4).
12 e(z,s(x3,x4)) | e(z,x3).
13 e(z,s(x3,x4)) | -e(z,x4).
14 R(x5,x5,x5).
15 -e($f3(x6),x6) | $ans(x6).
end_of_list.

% Heuristic:  Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).    weight(s(*1,s(*1,*1)),200).
weight(F(s(*1,*1),*1),200).    weight(F(*1,s(*1,*1)),200).
weight(F(F(*1,*1),*1),200).    weight(F(*1,F(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).            2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).   4  ($ans(s(x,x)) = $T).
5  ($ans(F(x,y)) = $T).        6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
10  e(x,F(y,z)) | -e(u,y) | -e(v,z) | -R(u,v,x).
```

```
12  e(x,s(y,z)) | e(x,y).
13  e(x,s(y,z)) | -e(x,z).
14  R(x,x,x).
15  -e($f3(x),x) | $ans(x).
17 [ur,12,15] e($f3(s(x,y)),x) | $ans(s(x,y)).
20 [ur,13,15] -e($f3(s(x,y)),y) | $ans(s(x,y)).
32 [ur,10,17,17,14] e($f3(s(x,y)),F(x,x)) | $ans(s(x,y)).
33 [binary,32,20] $ans(s(v64,F(v64,v64))).
------------ end of proof -------------
```

In set notation, the answer is $X \subseteq F(X,X)$.

Example d

(d) (i) F is lower semi-idempotent [$\forall X \subseteq U:\ F(X,X) \subseteq X$],
(ii) R is 3-prime [$\forall x,y,z\ \in U:\ Rxyz \Rightarrow z = x$ or $z = y$]

Direction (i) → (ii) (Top–Down)

Logical Formulation:	$\forall z\ z \in F(X,X) \Rightarrow z \in X$
Rewritten:	$\forall z\ (\exists x,y\ x \in X \wedge y \in Y \wedge R(x,y,z)) \Rightarrow z \in X$
Negated and Clausified:	$X(\underline{x})$
	$Y(\underline{y})$
	$R(\underline{x},\underline{y},\underline{z})$
	$\neg X(\underline{z})$
X 'Resolved Away':	$R(\underline{x},\underline{y},\underline{z})$
	$\underline{z} \neq \underline{x}$
	$\underline{z} \neq \underline{y}$
Quantifiers Reconstructed:	$\exists x,y,z\ R(x,y,z) \wedge z \neq x \wedge z \neq y$
Negated again:	$\forall x,y,z\ R(x,y,z) \Rightarrow (z = x \vee z = y)$

Direction (ii) → (i) (Bottom-Up)

Otter **Protocol:**

```
set(hyper_res).

formula_list(sos).
(all w all z all X all Y (e(w,z,F(X,Y)) <->
          (exists x exists y (e(w,x,X) & e(w,y,Y) & R(x,y,z))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x (x = x)).
(all x all y all z (R(x,y,z) -> ((z = x) | (z = y)))))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(sos).
7 -e(z,F(x1,x2)) | e($f2(z,x1,x2),x1).
8 -e(z,F(x1,x2)) | e($f1(z,x1,x2),x2).
9 -e(z,F(x1,x2)) | R($f2(z,x1,x2),$f1(z,x1,x2),z).
10 e(z,F(x1,x2)) | -e(x,x1) | -e(y,x2) | -R(x,y,z).
11 -e(z,s(x3,x4)) | -e(z,x3) | e(z,x4).
12 e(z,s(x3,x4)) | e(z,x3).
13 e(z,s(x3,x4)) | -e(z,x4).
14 (x = x).
15 -R(x5,x6,x7) | (x7 = x5) | (x7 = x6).
```

```
16 -e($f3(x8),x8) | $ans(x8).
end_of_list.

% Heuristic:  Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).    weight(s(*1,s(*1,*1)),200).
weight(F(s(*1,*1),*1),200).    weight(F(*1,s(*1,*1)),200).
weight(F(F(*1,*1),*1),200).    weight(F(*1,F(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).            2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).   4  ($ans(s(x,x)) = $T).
5  ($ans(F(x,y)) = $T).        6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
7  -e(x,F(y,z)) | e($f2(x,y,z),y).
8  -e(x,F(y,z)) | e($f1(x,y,z),z).
9  -e(x,F(y,z)) | R($f2(x,y,z),$f1(x,y,z),x).
11  -e(x,s(y,z)) | -e(x,y) | e(x,z).
12  e(x,s(y,z)) | e(x,y).
13  e(x,s(y,z)) | -e(x,z).
15  -R(x,y,z) | (z = x) | (z = y).
16  -e($f3(x),x) | $ans(x).
17 [hyper,12,16] e($f3(s(x,y)),x) | $ans(s(x,y)).
21 [hyper,13,12] e(x,s(y,z)) | e(x,s(z,u)).
22 [hyper,21,16] e($f3(s(x,y)),s(y,z)) | $ans(s(x,y)).
24 [hyper,7,17] e($f2($f3(s(F(x,y),z)),x,y),x) | $ans(s(F(x,y),z)).
26 [hyper,8,17] e($f1($f3(s(F(x,y),z)),x,y),y) | $ans(s(F(x,y),z)).
32 [hyper,9,17] R($f2($f3(s(F(x,y),z)),x,y),$f1($f3(s(F(x,y),z)),x,y),$f3(s(F(x,y),z))) |
         $ans(s(F(x,y),z)).
34 [hyper,32,15] $ans(s(F(x,y),z)) | ($f3(s(F(x,y),z)) = $f2($f3(s(F(x,y),z)),x,y)) |
         ($f3(s(F(x,y),z)) = $f1($f3(s(F(x,y),z)),x,y)).
191 [para_from,34,24] e($f3(s(F(x,y),z)),x) | $ans(s(F(x,y),z)) |
         ($f3(s(F(x,y),z)) = $f1($f3(s(F(x,y),z)),x,y)).
211 [hyper,191,11,22] $ans(s(F(x,y),x)) | ($f3(s(F(x,y),x)) = $f1($f3(s(F(x,y),x)),x,y)) |
         e($f3(s(F(x,y),x)),z).
224 [para_from,191,22] e($f1($f3(s(F(x,y),z)),x,y),s(z,u)) |
         $ans(s(F(x,y),z)) | e($f3(s(F(x,y),z)),x).
240 [hyper,224,11,22] e($f1($f3(s(F(x,y),x)),x,y),s(x,z)) |
         $ans(s(F(x,y),x)) | e($f3(s(F(x,y),x)),u).
246 [hyper,240,16] e($f1($f3(s(F(x,y),x)),x,y),s(x,z)) | $ans(s(F(x,y),x)).
251 [hyper,246,11,26] $ans(s(F(x,x),x)) | e($f1($f3(s(F(x,x),x)),x,x),y).
625 [hyper,211,16] $ans(s(F(x,y),x)) | ($f3(s(F(x,y),x)) = $f1($f3(s(F(x,y),x)),x,y)).
710 [para_from,625,16] -e($f1($f3(s(F(x,y),x)),x,y),s(F(x,y),x)) | $ans(s(F(x,y),x)).
711 [binary,710,251] $ans(s(F(v64,v64),v64)).
------------ end of proof -------------
```

In set notation, the answer is $F(X, X) \subseteq X$.

Example e

(e) (i) F is idempotent $[\forall X \subseteq U : \; F(X, X) = X]$,
(ii) R is totally reflexive and 3-prime

This result just combines the two previous ones.

Example f

(f) (i) F associates from left to right $[\forall X, Y, Z \subseteq U: \ F(F(X,Y),Z) \subseteq F(X, F(Y,Z))]$,
(ii) R^2 associates from left to right
$[\forall x, y, z, u \in U: \ (\exists v)[Rxyv \wedge Rvzu] \Rightarrow (\exists w)[Rxwu \wedge Ryzw]]$
Abbreviation: $[R^2(xy)zu \Rightarrow R^2x(yz)u]$

Direction (i) → (ii) (Top–Down)

Logical Formulation:

$$\forall z \ z \in F(F(X,Y),Z) \Rightarrow z \in F(F(X,Y),Z)$$

Rewritten:

$$\forall z \ (\exists x, y \ (\exists x_1, y_1 \ x_1 \in X \wedge y_1 \in Y \wedge R(x_1,y_1,x)) \wedge y \in Z \wedge R(x,y,z)) \Rightarrow$$
$$(\exists x, y \ x \in X \wedge (\exists x_1, y_1 \ x_1 \in X \wedge y_1 \in Y \wedge R(x_1,y_1,y)) \wedge R(x,y,z))$$

Since this example is quite complicated, we use it for demonstrating the application of the theorem prover for generating all the resolvents.

Otter **Protocol:**

```
set(binary_res).
set(factor).
set(print_kept).

formula_list(sos).
-(all z ((exists x (exists y ((exists x1 (exists y1
(X(x1) & Y(y1) & R(x1,y1,x)))) & Z(y) & R(x,y,z)))) ->
(exists x (exists y (X(x) &
(exists x1 (exists y1 (Y(x1) & Z(y1) & R(x1,y1,y)))) & R(x,y,z)))))).
end_of_list.

-------> sos clausifies to:
1  X($c2).
2  Y($c1).
3  R($c2,$c1,$c4).
4  Z($c3).
5  R($c4,$c3,$c5).
6  -X(x) | -Y(y) | -Z(z) | -R(y,z,u) | -R(x,u,$c5).
7 [factor,6] -X(x) | -Y(x) | -Z($c5) | -R(x,$c5,$c5).

** KEPT: 25 [binary,23,4] -R($c1,$c3,x) | -R($c2,x,$c5).
```

Clauses 3,5 and 25 are the only generated clauses consisting of R–literals only .
Quantifiers Reconstructed:

$$\exists c1, c2, c3, c4, c5 \ R(c2,c1,c4) \wedge R(c4,c3,c5) \wedge (\forall x \ \neg R(c1,c3,x) \vee \neg R(c2,x,c5))$$

Negated again:

$$\forall c1, c2, c3, c4, c5 \ (R(c2,c1,c4) \wedge R(c4,c3,c5)) \Rightarrow (\exists x \ R(c1,c3,x) \wedge R(c2,x,c5)),$$

or in a more readable notation:

$$\forall x, y, z, u \ (\exists v \ R(x,y,v) \wedge R(v,z,u)) \Rightarrow (\exists w \ R(x,w,u) \wedge R(y,z,w))$$

Direction (ii) → (i) (Bottom-Up)

Otter **Protocol:**

```
set(ur_res).
set(unit_deletion).

formula_list(sos).
(all w all z all X all Y (e(w,z,F(X,Y)) <->
         (exists x exists y (e(w,x,X) & e(w,y,Y) & R(x,y,z))))).
(all z all X all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))).
(all x all y all z all u ((exists v (R(x,y,v) & R(v,z,u)))
          -> (exists w (R(y,z,w) & R(x,w,u))))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(sos).
2 -e(z,F(x1,x2)) | e($f2(z,x1,x2),x1).
3 -e(z,F(x1,x2)) | e($f1(z,x1,x2),x2).
4 -e(z,F(x1,x2)) | R($f2(z,x1,x2),$f1(z,x1,x2),z).
5 e(z,F(x1,x2)) | -e(x,x1) | -e(y,x2) | -R(x,y,z).
6 -e(z,s(x3,x4)) | -e(z,x3) | e(z,x4).
7 e(z,s(x3,x4)) | e(z,x3).
8 e(z,s(x3,x4)) | -e(z,x4).
9 -R(x5,x6,x) | -R(x,x7,x8) | R(x6,x7,$f3(x5,x6,x7,x8)).
10 -R(x5,x6,x) | -R(x,x7,x8) | R(x5,$f3(x5,x6,x7,x8),x8).
11 -e($f4(x9),x9) | $ans(x9).
end_of_list.

% Heuristic:  Avoid double nestings of s and triple nestings of F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).       weight(s(*1,s(*1,*1)),200).
weight(F(*1,s(*1,*1)),200).       weight(F(s(*1,*1),*1),200).
weight(F(F(*1,F(*1,*1)),*1),200). weight(F(F(F(*1,*1),*1),*1),200).
weight(F(*1,F(F(*1,*1),*1)),200). weight(F(*1,F(*1,F(*1,*1))),200).
weight(F(F(*1,*1),F(*1,*1)),200). weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).
end_of_list.

----> UNIT CONFLICT at 265.00 sec ---->
-------------- PROOF ----------------
2  -e(x,F(y,z)) | e($f2(x,y,z),y).
3  -e(x,F(y,z)) | e($f1(x,y,z),z).
4  -e(x,F(y,z)) | R($f2(x,y,z),$f1(x,y,z),x).
5  e(x,F(y,z)) | -e(u,y) | -e(v,z) | -R(u,v,x).
7  e(x,s(y,z)) | e(x,y).
8  e(x,s(y,z)) | -e(x,z).
9  -R(x,y,z) | -R(z,u,v) | R(y,u,$f3(x,y,u,v)).
10  -R(x,y,z) | -R(z,u,v) | R(x,$f3(x,y,u,v),v).
11  -e($f4(x),x) | $ans(x).
13 [ur,7,11] e($f4(s(x,y)),x) | $ans(s(x,y)).
16 [ur,8,11] -e($f4(s(x,y)),y) | $ans(s(x,y)).
21 [ur,2,13] e($f2($f4(s(F(x,y),z)),x,y),x) | $ans(s(F(x,y),z)).
22 [ur,3,13] e($f1($f4(s(F(x,y),z)),x,y),y) | $ans(s(F(x,y),z)).
24 [ur,21,3] $ans(s(F(F(x,y),z),u)) | e($f1($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),y).
25 [ur,21,2] $ans(s(F(F(x,y),z),u)) |
            e($f2($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),x).
30 [ur,4,21] R($f2($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),
                        $f1($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),
                        $f2($f4(s(F(F(x,y),z),u)),F(x,y),z)) |
            $ans(s(F(F(x,y),z),u)).
```

```
31 [ur,4,13] R($f2($f4(s(F(x,y),z)),x,y),$f1($f4(s(F(x,y),z)),x,y),$f4(s(F(x,y),z))) |
            $ans(s(F(x,y),z)).
263 [ur,30,10,31] $ans(s(F(F(x,y),z),u)) | R($f2($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),
            $f3($f2($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),
            $f1($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),
            $f1($f4(s(F(F(x,y),z),u)),F(x,y),z),$f4(s(F(F(x,y),z),u))),
            $f4(s(F(F(x,y),z),u))).
264 [ur,30,9,31] $ans(s(F(F(x,y),z),u)) |
            R($f1($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),$f1($f4(s(F(F(x,y),z),u)),
            F(x,y),z),$f3($f2($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),
            $f1($f2($f4(s(F(F(x,y),z),u)),F(x,y),z),x,y),
            $f1($f4(s(F(F(x,y),z),u)),F(x,y),z),$f4(s(F(F(x,y),z),u)))).
279 [ur,263,5,16,25] $ans(s(F(F(x,y),z),F(x,u))) |
            -e($f3($f2($f2($f4(s(F(F(x,y),z),F(x,u))),F(x,y),z),x,y),
            $f1($f2($f4(s(F(F(x,y),z),F(x,u))),F(x,y),z),x,y),
            $f1($f4(s(F(F(x,y),z),F(x,u))),F(x,y),z),$f4(s(F(F(x,y),z),F(x,u)))),u).
306 [ur,279,5,24,22] $ans(s(F(F(x,y),z),F(x,F(u,v)))) |
            -R($f1($f2($f4(s(F(F(w,u),v6),v7)),F(w,u),v6),w,u),
            $f1($f4(s(F(v8,v),v9)),v8,v),$f3($f2($f2($f4(s(F(F(x,y),z),
            F(x,F(u,v)))),F(x,y),z),x,y),$f1($f2($f4(s(F(F(x,y),z),F(x,F(u,v)))),
            F(x,y),z),x,y),$f1($f4(s(F(F(x,y),z),F(x,F(u,v)))),F(x,y),z),
            $f4(s(F(F(x,y),z),F(x,F(u,v)))))) |
            $ans(s(F(F(w,u),v6),v7)) | $ans(s(F(v8,v),v9)).
307 [binary,306,264] $ans(s(F(F(v64,v65),v66),F(v64,F(v65,v66)))).
------------ end of proof -------------
```

In set notation, the answer is $F(F(X,Y),Z) \subseteq F(X,F(Y,Z))$.

Example g

(g) (i) F associates from right to left $[\forall X, Y, Z \ \subseteq U: \ F(X,F(Y,Z)) \subseteq F(F(X,Y),Z)]$,
(ii) R^2 associates from right to left
$[\forall x, y, z, u, \ \in U: \ (\exists v)[Rxvu \wedge Ryzv] \Rightarrow (\exists w)[Rxyw \wedge Rwzu]]$
Abbreviation: $[R^2x(yz)u \Rightarrow R^2(xy)zu]$

This case is symmetric to the previous one.

Example h

(h) (i) F is associative,
(ii) R^2 is associative $[R^2(xy)zu$ iff $R^2x(yz)u]$

This case combines the two previous results.

Example i

(i) (i) $E \subseteq U$ is a left identity of F $[\forall X \subseteq U: \ F(E,X) = X]$,
(ii) $E \subseteq U$ is a set of left identities of R $[\forall x, y \subseteq U: \ (\exists e \in E)[Rexy]$ iff $x = y]$.

It is again advantageous to split the equation $F(E,X) = X$ into the two cases $F(E,X) \subseteq X$ and $F(E,X) \supseteq X$.

Direction (i) → (ii) (Top–Down)
'⊆–Part

Logical Formulation:	$\forall y \ y \in F(E,X) \Rightarrow y \in X$
Rewritten:	$\forall y \ (\exists e, x \ e \in E \wedge x \in X \wedge R(e,x,y)) \Rightarrow y \in X$
Negated and Clausified:	$E(\underline{e})$
	$X(\underline{x})$

	$R(\underline{e}, \underline{a}, \underline{y})$
	$\neg X(\underline{y})$
X 'Resolved Away':	$E(\underline{e})$
	$R(\underline{e}, \underline{a}, \underline{y})$
	$\underline{e} \neq \underline{y}$
Quantifiers Reconstructed:	$\exists x, y, e\ E(e) \wedge R(e, x, y) \wedge x \neq y)$
Negated again:	$\forall x, y\ (\exists e\ E(e) \wedge R(e, x, y)) \Rightarrow x = y)$

'⊇–Part

Logical Formulation:	$\forall y\ y \in X \Rightarrow y \in F(E, X)$
Rewritten:	$\forall y\ y \in X \Rightarrow (\exists e, x\ e \in E \wedge x \in X \wedge R(e, x, y))$
Negated and Clausified:	$X(\underline{y})$
	$\neg E(e), \neg X(x), \neg R(e, x, \underline{y})$
X 'Resolved Away':	$\neg E(e), \neg R(e, \underline{y}, \underline{y})$
Quantifiers Reconstructed:	$\exists y\ \forall e\ \neg E(e) \vee \neg R(e, y, y)$
Negated again:	$\forall y\ \exists e\ E(e) \wedge R(e, y, y))$

Both parts together yield $\forall x, y\ (\exists e\ E(e) \wedge R(e, x, y)) \Leftrightarrow x = y$.

Direction (ii) → (i) (Bottom-Up)

'⊆–Part

Otter **Protocol:**

```
set(ur_res).

formula_list(sos).
(all w all z all X all Y (e(w,z,F(X,Y)) <->
          (exists x exists y (e(w,x,X) & e(w,y,Y) & R(x,y,z))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x (x = x)).
(all x all y ((exists z (e(z,E) & R(z,x,y))) <-> (x = y))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(sos).
7  -e(x,F(y,z)) | e($f2(x,y,z),y).
8  -e(x,F(y,z)) | e($f1(x,y,z),z).
9  -e(x,F(y,z)) | R($f2(x,y,z),$f1(x,y,z),x).
10  e(x,F(y,z)) | -e(u,y) | -e(v,z) | -R(u,v,x).
11  -e(x,s(y,z)) | -e(x,y) | e(x,z).
12  e(x,s(y,z)) | e(x,y).
13  e(x,s(y,z)) | -e(x,z).
14  (x = x).
15  -e(x,E) | -R(x,y,z) | (y = z).
16  e($f3(x,y),E) | (x != y).
17  R($f3(x,y),x,y) | (x != y).
18  -e($f4(x),x) | $ans(x).end_of_list.

% Heuristic:  Avoid double nestings of s and F.
weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).   weight(s(*1,s(*1,*1)),200).
weight(F(s(*1,*1),*1),200).   weight(F(*1,s(*1,*1)),200).
weight(F(F(*1,*1),*1),200).   weight(F(*1,F(*1,*1)),200).
weight(junk,200).
end_of_list.

list(demodulators).
1  (s(x,x) = junk).           2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).  4  ($ans(s(x,x)) = $T).
5  ($ans(F(x,y)) = $T).       6  ($ans(junk) = $T).
end_of_list.

---------------- PROOF ----------------
10  e(x,F(y,z)) | -e(u,y) | -e(v,z) | -R(u,v,x).
12  e(x,s(y,z)) | e(x,y).
13  e(x,s(y,z)) | -e(x,z).
14  (x = x).
16  e($f3(x,y),E) | (x != y).
17  R($f3(x,y),x,y) | (x != y).
18  -e($f4(x),x) | $ans(x).
20 [ur,12,18] e($f4(s(x,y)),x) | $ans(s(x,y)).
23 [ur,13,18] -e($f4(s(x,y)),y) | $ans(s(x,y)).
28 [ur,16,14] e($f3(x,x),E).
30 [ur,17,14] R($f3(x,x),x,x).
39 [ur,10,28,20,30] e($f4(s(x,y)),F(E,x)) | $ans(s(x,y)).
40 [binary,39,23] $ans(s(v64,F(E,v64))).
------------ end of proof -------------
```

In set notation, the answer is $X \subseteq F(E, X)$.

'$\supseteq$–Part

```
set(ur_res).

formula_list(sos).
(all w all z all X all Y (e(w,z,F(X,Y)) <->
          (exists x exists y (e(w,x,X) & e(w,y,Y) & R(x,y,z))))).
(all z (all X (all Y (e(z,s(X,Y)) <-> (e(z,X) -> e(z,Y)))))).
(all x (x = x)).
(all x all y ((exists z (e(z,E) & R(z,x,y))) -> (x = y)))).
-(exists f (all z (e(z,f) & -$ans(f)))).
end_of_list.

Clauses:
list(sos).
7  -e(x,F(y,z)) | e($f2(x,y,z),y).     8  -e(x,F(y,z)) | e($f1(x,y,z),z).
9  -e(x,F(y,z)) | R($f2(x,y,z),$f1(x,y,z),x).
10  e(x,F(y,z)) | -e(u,y) | -e(v,z) | -R(u,v,x).
11  -e(x,s(y,z)) | -e(x,y) | e(x,z).  12  e(x,s(y,z)) | e(x,y).
13  e(x,s(y,z)) | -e(x,z).            14  (x = x).
15  -e(x,E) | -R(x,y,z) | (y = z).    16  -e($f3(x),x) | $ans(x).
end_of_list.

weight_list(purge_gen).
weight(s(s(*1,*1),*1),200).          weight(s(*1,s(*1,*1)),200).
weight(F(s(*1,*1),*1),200).          weight(F(*1,s(*1,*1)),200).
weight(F(F(*1,*1),*1),200).          weight(F(*1,F(*1,*1)),200).
weight(junk,200).
end_of_list.
```

```
list(demodulators).
1  (s(x,x) = junk).                  2  ($ans(s(s(x,y),z)) = $T).
3  ($ans(s(x,s(y,z))) = $T).         4  ($ans(s(x,x)) = $T).
5  ($ans(F(x,y)) = $T).              6  ($ans(junk) = $T).
end_of_list.
---------------- PROOF ----------------
7  -e(x,F(y,z)) | e($f2(x,y,z),y).
8  -e(x,F(y,z)) | e($f1(x,y,z),z).
9  -e(x,F(y,z)) | R($f2(x,y,z),$f1(x,y,z),x).
12  e(x,s(y,z)) | e(x,y).
13  e(x,s(y,z)) | -e(x,z).
15  -e(x,E) | -R(x,y,z) | (y = z).
16  -e($f3(x),x) | $ans(x).
18 [ur,12,16] e($f3(s(x,y)),x) | $ans(s(x,y)).
26 [ur,7,18] e($f2($f3(s(F(x,y),z)),x,y),x) | $ans(s(F(x,y),z)).
27 [ur,8,18] e($f1($f3(s(F(x,y),z)),x,y),y) | $ans(s(F(x,y),z)).
29 [ur,27,13] $ans(s(F(x,y),z)) | e($f1($f3(s(F(x,y),z)),x,y),s(u,y)).
30 [ur,9,18] R($f2($f3(s(F(x,y),z)),x,y),$f1($f3(s(F(x,y),z)),x,y),$f3(s(F(x,y),z))) |
          $ans(s(F(x,y),z)).
41 [ur,30,15,26] $ans(s(F(E,x),y)) | ($f1($f3(s(F(E,x),y)),E,x) = $f3(s(F(E,x),y))).
70 [para_from,41,29] $ans(s(F(E,x),y)) | e($f3(s(F(E,x),y)),s(z,x)).
74 [binary,70,16] $ans(s(F(E,y),y)).
------------ end of proof -------------
```

In set notation, the answer is $F(E,Y) \subseteq Y$.

9 Summary

We have shown how corresponding properties of relations in a structure and the corresponding functions in the power structure can be computed automatically in both directions. In the direction from power structures to structures we used a quantifier elimination method whereas in the other direction an automated guess and verify method was proposed. The guessing part, however, could also be automated using a theorem prover which can enumerate proofs.

The duality problem for power structures is prototypical for many other applications, in particular for computing the correspondences between an axiomatic description of a logic by means of an Hilbert calculus and its model theoretic semantics. The examples we have investigated in this paper can be transferred directly to modal logic. Figuring out these correspondences is the key for developing efficient calculi for new logics.

As long as there is no special implementation of the algorithms (which is not complicated, but takes time), we would like to encourage the reader to experiment with the OTTER theorem prover. It has been ported to many machines and is easily available.

References

[AB75] A.R. Anderson and N.D. Belnap. *Entailment: The Logic of Relevance and Necessity.* Princeton University Press, 1975.

[Abr87] S. Abramsky. Domain theory in logical form. In *Proceedings, Symposium on Logic in Computer Science*, pages 47–53, Ithaca, NY, 1987.

[Ack35a] Wilhelm Ackermann. Untersuchung über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.

[Ack35b] Wilhelm Ackermann. Zum Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 111:61–63, 1935.

[Ack54] Wilhelm Ackermann. *Solvable Cases of the Decision Problem.* North–Holland Pu. Co., 1954.

[BB91] Karl-Hans Bläsius and Hans-Jürgen Bürckert. *Deduction Systems in Artificial Intelligence.* Ellis Horwood Series in Artificial Intelligence, 1991.

[BGW92] Leo Bachmair, Harald Ganzinger and Uwe Waldmann. Theorem proving for hierarchic first-order theories, 1992. To appear in Proc. ALP'92, Lecture Notes in Comp. Science.

[BP89] W.J. Blok and D. Pigozzi. Algebraizable logics. *Memoirs of the American Mathematical Society*, 77(396):1–78, January 1989.

[Bri89] C. Brink. $\mathbf{R}^{\neg}$-algebras and $\mathbf{R}^{\neg}$-model structures as power constructs. *Studia Logica*, 48:85-109, 1989.

[Bri92] C. Brink. Power structures. *Algebra Universalis*, 1992. To appear.

[BS84] R. Bull and K. Segerberg. Basic modal logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 1–88. Reidel, Dordrecht, Holland, 1984.

[BS85] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[BS92] C. Brink and R.A. Schmidt. Subsumption computed algebraically. *Computers and Mathematics with Applications*, 23:329–342, 1992. Special Issue on Semantic Networks in Artificial Intelligence.

[CL73] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science and Applied Mathematics Series. Academic Press, New York, 1973.

[Cra74] W. Craig. *Logic in algebraizable form*, volume 72 of *Studies in Logic and Foundations of Mathematics.* North-Holland, 1974.

[Gin88] M.L. Ginsberg. Multivalued logics: a uniform apporoach to reasoning in artificial intelligence. *Computing Intelligence*, 4:265–316, 1988.

[GO92] Dov M. Gabbay and Hans Jürgen Ohlbach. Quantifier elimination in second–order predicate logic. *South African Computer Journal*, 7:35–43, July 1992. Appeared also in Proc. of KR92, Morgan Kaufmann, 1992, pages 425–436.

[Hal62] P.R. Halmos. *Algebraic Logic.* Chelsea, New York, 1962.

[Han83] G. Hansoul. A duality for Boolean algebras with operators. *Algebra Universalis*, 17:34–49, 1983.

[Hen61] L. Henkin. Some remarks on infinitely long formulas. In *Infinitistic Methods*, pages 167–183. Pergamon Press, Oxford, 1961.

[HJ87] C.A.R. Hoare and He Jifeng. The weakest prespecification. *Information Processing Letters*, 24:127–132, 1987.

[HMT71] L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras I*, volume 64 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, 1971.

[HMT85] L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras II*, volume 115 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, 1985.

[Joh82] P.T. Johnstone. *Stone spaces.* Cambridge studies in advanced mathematics. Cambridge University Press, 1982.

[Jón] B. Jónsson. Program specifications as Boolean operators: A very preliminary draft. Department of Mathematics, Vanderbilt University, Nashville, TN.

[Jón92] B. Jónsson. A survey of Boolean algebras with operators. Congress of Young Mathematicians, Montreal, 1991.

[JT51] B. Jónsson and A. Tarski. Boolean algebras with operators, Part I. *American Journal of Mathematics*, 73:891–939, 1951.

[JT52] B. Jónsson and A. Tarski. Boolean algebras with operators, Part II. *American Journal of Mathematics*, 74:127–162, 1952.

[Koz81] D. Kozen. On the duality of dynamic algebras and Kripke models. In E. Engeler, editor, *Logic of Programs*, volume 125 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1981.

[Kri59] S.A. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–14, 1959.

[LO84] Ewing Lusk and Ross Overbeek. The automated reasoning system ITP. Tech. Report ANL-84/27, Argonne National Laboratory, Argonne, Ill., April 1984.

[Lov78] Don Loveland. *Automated Theorem Proving: A Logical Basis.*, volume 6 of *Fundamental Studies in Computer Science.* North-Holland, New York, 1978.

[McC89] William W. McCune. *OTTER User's Guide.* Mathematical and Computer Science Division, Argonne National Laboratory, april 1989.

[Nem92] I. Nemeti. Algebraizations of quantifier logics, an introductory overview. *Studia Logica*, to appear.

[MOW76] John D. McCharen, Ross Overbeek, and Lawrence Wos. Complexity and related enhancements for automated theorem-proving programs. *Computers and Mathematics with Applications*, 2:1–16, 1976.

[OS91] Hans Jürgen Ohlbach and Jörg H. Siekmann. The Markgraf Karl refutation procedure. In Jean Luis Lassez and Gordon Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson*, pages 41–112. MIT Press, 1991.

[Pra90] V.R. Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In C.H. Bergman, R.D. Maddux, and D.L. Pigozzi, editors, *Algebraic Logic and Universal Algebra in Computer Science*, volume 425 of *Lecture Notes in Computer Science*, pages 77–110. Springer-Verlag, 1990.

[Pri70] H.A. Priestley. Representation of distributive lattices by means of ordered Stone spaces. *Bulletin of the London Mathematical Society*, 2:186–190, 1970.

[Rob65a] John Alan Robinson. Automated deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1(3):227–234, 1965.

[Rob65b] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery (JACM)*, 12(1):23–41, 1965.

[RW69] G. Robinson and Larry Wos. *Machine Intelligence*, volume 4, chapter Paramodulation and TP in First Order Theories with Equality., pages 135–150. Edinburgh University Press, Edinburgh, 1969.

[Sch91] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI 91*, pages 466–471. Morgan Kaufmann, 1991.

[Sim93] Harold Simmons. The Monotonous Elimination of Predicate Variables. Forthcoming in Journal of Logic and Computation, 1993.

[Smi88] Brian Smith. Reference manual for the environmental theorem prover: An incarnation of AURA. Tech. Report ANL-88-2, Argonne National Laboratory, Argonne, Ill., 1988.

[Sto37] M.H. Stone. The theory of representations for Boolean algebras. *American Mathematical Society*, 6:37–111, 1937.

[Sza92] Andrzej Szałas. On correspondence between modal and classical logic: Automated approach. Technical Report MPI–I–92–209, Max Planck Institut für Informatik, Saarbrücken, March 1992.

[Tar41] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.

[TG87] Alfred Tarski and Steven Givant. *A Formalization of Set Theory without Variables.* American Mathematical Society Colloquium Publications 41, Providence, Rhode Island, 1987.

[vB84] Johan van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 167–248. Reidel, Dordrecht, Holland, 1984.

[WOLB91] Larry Wos, Ross Overbeek, Ewing Lusk and Jim Boyle. *Automated Reasoning. Introduction and Applications*, Second Edition, McGraw–Hill Inc., 1991.