A Rewriting Decision Procedure for Dijkstra-Scholten's Syllogistic Logic with Complements

Camilo Rocha
José Meseguer

December, 2007

Technical Report
UIUCDCS-R-2007-2921

Formal Methods and Declarative Languages Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N Goodwin Ave
Urbana, IL 61801

The formalist, however, prefers to manipulate his formulae,
temporarily ignoring all interpretations they might admit, the rules
for the permissible symbol manipulations being formulated
in terms of those symbols: the formalist calculates
with uninterpreted formulae.

E.W. Dijkstra, “The notational conventions I adopted, and why”
EWD 1300.

Table of Contents

Abstract IV

1 Introduction 1

2 Preliminaries 3

3 Equational Axioms of DS Propositional Logic 5

4 Specification in Maude 10

5 A Decision procedure for Syllogistic Logic with Complements . . 13

5.1 Syllogistic Logic with Complement and Boolean Connectives 15

5.2 The equational theory T_{CSYLL} 17

5.3 Examples 23

6 Conclusions 25

List of Figures

1 Confluence of $\overset{*}{\rightarrow}_{E/A}$ 5

Abstract

We present an equational decision procedure *à la* Dijkstra & Scholten for the 'Syllogistic Logic with Complements'. First, we give an equational axiomatization of Dijkstra & Scholten's propositional logic and show how it gives a decision procedure for propositional logic by equational term rewriting. We also show how one can efficiently obtain certain propositional models (i.e., truth assignments over propositional variables) from the canonical forms given by the system. We then present the Syllogistic Logic with Complements and show how the decision procedure for propositional logic can be easily extended to obtain a decision procedure for this subset of First-Order Logic. Moreover, an executable specification of both decision procedures is presented in the Maude system, and examples illustrating the use of both canonical equational systems are given.

1 Introduction

This paper illustrates some simple, yet nontrivial, uses of equational logic as a logical framework. Specifically, we show how axiomatizations of both the propositional calculus of E.W. Dijkstra and C.S. Scholten and the Syllogistic Logic with Complements with Boolean connectives of L.S. Moss [23] can be represented in the equational logic framework yielding decision procedures for both logics.

In 1990 E.W. Dijkstra and C.S. Scholten [9] presented a calculus for predicate transformers suited for verifying and deriving programs, together with the 'calculational style': an homogenous and compact proof format in which formal syntax was used to develop proofs while annotating logical deductions with text explanations. The calculus for predicate transformers is based upon an axiomatization of Boolean expressions that facilitates and guides proofs. Within the five years following that work, D. Gries and F. Schneider presented a teaching methodology in discrete mathematics using this calculational style [13] and gave a detailed proof of soundness of the system for propositional logic [14]. In the late 1990's V. Lifschitz gave a detailed logical basis to the proof format used in the system [20]. Besides the mechanization by rewriting presented in Section 3, we are not aware of any other mechanizations of the Dijkstra-Scholten logic. Therefore, this work, together with the technical report [25], that further develops the propositional aspects, may also be of interest for applications, such as program verification, in which the Dijkstra-Scholten logic plays a crucial role. Likewise, we are not aware of any mechanization of Moss' Syllogistic Logic with Complements and Boolean connectives. Therefore, the decision procedure presented in this work and implemented in Maude seems to be the first mechanization that has been developed.

A *syllogism* consists of three syllogistic propositions, two premises (or hypotheses) and one conclusion (or thesis). A *syllogistic proposition* is a sentence affirming or denying a fact of the form $\mathbf{A}PQ$ (All P are Q), $\mathbf{E}PQ$ (No P is Q), $\mathbf{I}PQ$ (Some P is Q) and $\mathbf{O}PQ$ (Some P are not Q). A syllogism is considered *valid* if its thesis follows from the two hypotheses by applying the Aristotelian deduction rules which, from the logical point of view, are particular instances of those of First-Order Logic[1] (FOL). For example, 'All men are mortal and all Colombians are men, then all Colombians are mortal' is a valid syllogism. It was G. Frege in the late XIX century who pointed out the need for a more general logical system in or-

[1] Currently, this is widely accepted from the mathematical point of view, but somewhat disputed from a philosophic point of view. We refer the reader to [11] for further details.

der to express the foundations of all deductive reasoning. Since then, FOL has been the main subject of study for logicians interested in this area. In the past few years, the interest in Syllogistic Logic has experienced a special awakening as ‘specialists in communication and information theory employ ideas which can be traced back to Aristotle’s work’ [11]. A novel approach has been proposed by L.S. Moss in his axiomatization of the Syllogistic Logic with Complements (CSYLL) where Boolean connectives and the complement relation are allowed in syllogisms, thus achieving a more general logic than the traditional syllogistic one [23]. Our main goal in this paper is to provide an equational-based decision procedure for CSYLL which we obtain by extending our Dijkstra-Scholten decision procedure for propositional logic.

The general idea of our approach is as follows. We view equational logic as a *computational logic*, that is, as a logic which can be used as a programming language and in which computation and deduction coincide. For this to work in practice, the equational theories used as programs should satisfy appropriate executability requirements, so that their equations can be used as *simplification rules* from left to right. Typical such requirements are that the rules should be *terminating* (i.e., that there are no infinite simplification sequences) and *confluent*, which under the termination assumption is equivalent to the fact that each expression can be fully simplified to a *unique* equivalent expression, called its *canonical form*, that cannot be further simplified. For example, in a suitable equational axiomatization of arithmetic $2 + 2$ is simplified to the canonical form 4. Similarly, as we shall see later, in a suitable equational axiomatization of the propositional calculus the proposition $p \vee \neg p$ has canonical form T.

Not only is equational logic itself a computational logic. It is also what is called a *logical framework*, that is, a meta-logic in which we can represent and axiomatize other logics. Furthermore, since equational logic is *reflective*, that is, since it can correctly encode its own meta-level [7], it is also a meta-logical framework [2], that is a logic in which we can both represent other logics and reason formally about the meta-logical properties of the logics thus represented. In this paper we will not pursue the meta-logical framework aspects of equational logic: we will just focus on its logical framework uses for the Dijkstra-Scholten logic and for CSYLL. However, the meta-logical framework aspect is also relevant, since we could use inductive theorem proving in equational logic and reflection to investigate meta-logical properties of both the Dijkstra-Scholten logic and of CSYLL.

Any computational logic worth its salt should be supported by efficient implementations. In this paper we use Maude [5, 6], which is one of the most efficient and general implementations of equational logic, and of a superlogic of equational logic called rewriting logic [22]. It turns out that Maude is particularly well-suited for the mechanization of the Dijkstra-Scholten logic and CSYLL because it efficiently supports not just equational simplification, but also equational simplification *modulo* a set of axioms, such as associativity and commutativity, that can be 'built in' for simplification purposes. We can illustrare this idea with a simple example. As it is well-known, the logical disjunction operator $\vee$ is associative and commutative. Suppose that we want to simplify propositional expressions using equations such as $p \vee \neg p = \mathsf{T}$, and $p \vee \mathsf{T} = \mathsf{T}$. Syntactically speaking, we cannot apply any of these equations to the proposition $(p \vee q) \vee \neg p$. However, if we use associativity and commutativity in a built-in way to rearrange our expression into its equivalent form $(p \vee \neg p) \vee q$, we can simplify it with the first equation to $\mathsf{T} \vee q$ and, using commutativity in a built-in way, we can then further simplify it to T with the second equation.

The paper is organized as follows. Notational conventions and basic definitions are given in Section 2. In Section 3 we exhibit our axiomatization of the propositional logic of Dijkstra & Scholten and show that its equational representation gives a decision procedure for propositional logic by equational term rewriting. Section 4 presents the corresponding executable specification in Maude of such decision procedure. An equational theory for CSYLL, shown to provide a decision procedure for CSYLL based on rewriting, is presented in Section 5, together with its executable specification in Maude and some examples. Finally, in Section 6 we present some conclusions.

2 Preliminaries

Throughout this paper we use t, u, v, t' , t'' and t_i to denote terms, p_i to denote propositional variables. Sometimes lower case letters are also used as quantified variables in FOL formulas.

Generally, upper case letters such as E and A denote sets, while the upper case letter T is reserved for theory names. The greek letter Σ, perhaps decorated with subscripts, is used to denote a signature of function symbols. Upper case letters, such as P, Q and R, denote variables.

Theories in *equational logic* are called *equational theories*. An *equational theory* is a pair (Σ, E), where Σ, called the signature, describes the syntax of the theory, that is, what *sorts* (types of data) and what *function symbols* (operation symbols) are involved, and E is a set of equations over *terms* (or expressions) in the syntax of Σ. Equational logic is a special case of a more general logic called membership equational [24, 4] in which, in addition to equations of the form $t = t'$, membership predicates of the form $t : s$, stating that term t has sort s, are also atomic sentences.

The *models* of an equational theory $T = (\Sigma, E)$ are *algebras.* Among these models a special role is played by the *initial algebra* of T, which is denoted by $T_{\Sigma/E}$, whose elements are equivalence classes of *ground* Σ-terms (terms without variables) modulo provable equality. An equality $t = t'$ is *provable* from T, written $T \vdash t = t'$, iff it can be deduced from the equations E by the rules of equational deduction.

Let $T = (\Sigma, E \cup A)$ be an equational theory. We assume that the set of equations of T is decomposed into a disjoint union $E \uplus A$, where A is a set of equations such as associativity, commutativity and identity of some function symbols in Σ, which are used as 'structural axioms', and E is a set of equations used as 'simplification rules' or 'rewrite rules' *modulo* the structural axioms A. Intuitively, we want to raise the level of abstraction by reasoning equationally with the equations E not just on terms t, but on equivalence classes $[t]_A$ modulo the structural axioms A. We say that the equations $t = t' \in E$ are admissible as *equational simplification rules* modulo A if any variable mentioned in t' is mentioned in t. If the equations in T are admissible as simplification rules, we can use them from left to right to (hopefully) bring terms to a simpler form which can be interpreted as their *evaluation.* This simplification process is called *equational simplification* or *rewriting* and gives rise to binary relations $\rightarrow_{E/A}$ and $\overset{*}{\rightarrow}_{E/A}$ over A-equivalence classes in $T_{\Sigma/A}$. The relation $\overset{*}{\rightarrow}_{E/A}$ is the reflexive and transitive closure of $\rightarrow_{E/A}$. Specifically, the relation $\rightarrow_{E/A}$ is defined as follows: $[t]_A \rightarrow_{E/A} [t']_A$ iff there is a representative $u \in [t]_A$ such that u can be decomposed as $u = C[v]$, with C a context and v a subterm of u, and there is an equation $t = t' \in E$ and a substitution σ such that (i) $v = \sigma(t)$, and (ii) $C[\sigma(t')] \in [t']_A$.

We say that $\rightarrow_{E/A}$ is *terminating* if there is no infinite sequence

$$[t_0]_A \rightarrow_{E/A} [t_1]_A \rightarrow_{E/A} \cdots [t_k]_A \rightarrow_{E/A} [t_{k+1}]_A \rightarrow_{E/A} \cdots .$$

We say that $\rightarrow_{E/A}$ is *confluent* if whenever we have $[t]_A \overset{*}{\rightarrow}_{E/A} [t']_A$ and $[t]_A \overset{*}{\rightarrow}_{E/A} [t'']_A$, there is a $[u]_A$ such that $[t']_A \overset{*}{\rightarrow}_{E/A} [u]_A$ and $[t'']_A \overset{*}{\rightarrow}_{E/A} [u]_A$. Pictorially, this is represented in Figure 1.

An equivalence class $[t]_A$ is called an E/A-*canonical form* if there is no $[t']_A$ such that $[t]_A \rightarrow_{E/A} [t']_A$. If the theory $T = (\Sigma, E \uplus A)$ is such that

$$\begin{tikzcd}
 & {[t]_A} \arrow[dl, "*", "E/A"'] \arrow[dr, "*"', "E/A"] & \\
{[t']_A} \arrow[dr, dashed, "*", "E/A"'] & & {[t'']_A} \arrow[dl, dashed, "*"', "E/A"] \\
 & {[u]_A} &
\end{tikzcd}$$

Fig. 1. Confluence of $\overset{*}{\rightarrow}_{E/A}$

the equations E are terminating and confluent modulo A, then each equivalence class $[t]_A$ can be rewritten to a *unique* canonical form $\mathsf{can}_{E/A}[t]$. This gives us an efficient *decision procedure* for equational reasoning in T, namely,

$$T \vdash t = t' \quad \text{iff} \quad \mathsf{can}_{E/A}[t] = \mathsf{can}_{E/A}[t'] \, .$$

3 Equational Axioms of DS Propositional Logic

The first axiomatization of DS propositional logic was presented by Dijkstra & Scholten [9], together with a calculus of predicates specialized for verifying and deriving programs from specifications. Gries & Schneider [13, 14] gave a complete formalization of the calculus. We present below our structured equational axiomatization of the DS propositional calculus in which we make an explicit separation between axioms A_{DS} to be used as 'structural axioms', and axioms E_{DS} that should be used as 'simplification rules' *modulo* A_{DS}.

Definition 1. *The equational theory* $T_{\mathrm{DS}} = (\Sigma_{\mathrm{DS}}, E_{\mathrm{DS}} \uplus A_{\mathrm{DS}})$ *of DS propositional logic is given by:*

$$\Sigma_{\mathrm{DS}} = \{\mathsf{F}^{(0)}, \mathsf{T}^{(0)}, \neg^{(1)}, \equiv^{(2)}, \not\equiv^{(2)}, \vee^{(2)}, \wedge^{(2)}, \Rightarrow^{(2)}, \Leftarrow^{(2)}\} \ ,$$

$$\begin{aligned}
A_{\mathrm{DS}} = \{ & P \equiv (Q \equiv R) \;=\; (P \equiv Q) \equiv R \, , \; P \equiv Q \;=\; Q \equiv P \, , \\
& P \not\equiv (Q \not\equiv R) \;=\; (P \not\equiv Q) \not\equiv R \, , \; P \not\equiv Q \;=\; Q \not\equiv P \, , \\
& P \vee (Q \vee R) \;=\; (P \vee Q) \vee R \, , \; P \vee Q \;=\; Q \vee P \, , \\
& P \wedge (Q \wedge R) \;=\; (P \wedge Q) \wedge R \, , \; P \wedge Q \;=\; Q \wedge R\} \ ,
\end{aligned}$$

$$\begin{aligned} E_{\mathrm{DS}} = \{ & P \equiv P \;=\; \mathsf{T}\,,\; P \equiv \mathsf{T} \;=\; P\,,\; \neg P \;=\; P \equiv \mathsf{F}\,,\\ & P \not\equiv Q \;=\; P \equiv Q \equiv \mathsf{F}\,,\; P \vee \mathsf{F} \;=\; P\,,\; P \vee \mathsf{T} \;=\; \mathsf{T}\,,\; P \vee P \;=\; P\,,\\ & P \vee (Q \equiv R) \;=\; (P \vee Q) \equiv (P \vee R)\,,\; P \wedge Q \;=\; P \equiv Q \equiv (P \vee Q)\,,\\ & P \Rightarrow Q \;=\; Q \equiv P \vee Q\,,\; P \Leftarrow Q \;=\; P \equiv P \vee Q\}, \end{aligned}$$

where P, Q and R are Σ_{DS}-variables.

The constants F and T denote the obvious false and true constant symbols. The rest of function symbols in Σ_{DS} have the following intended meaning and are listed by decreasing binding power: $\neg$ denotes negation, $\vee$ and $\wedge$ denote disjunction and conjunction respectively, $\Rightarrow$ and $\Leftarrow$ denote implication and consequence respectively, $\equiv$ and $\not\equiv$ denote equivalence and discrepancy respectively.

The axioms in A_{DS} express the associativity and commutativity properties (AC) of some binary operators. The set of axioms E_{DS}, to be used as simplification rules modulo A_{DS}, indicate that equivalence is reflexive and has identity T, that discrepancy and negation can be defined in terms of equivalence and the constant F, that disjunction is idempotent and distributes over equivalence while having identity F and annihilator T, that conjunction can be expressed in terms of equivalence and negation using the *golden rule* [13], that implication has the traditional meaning and, finally, that consequence can be treated as an implication by swapping its operands.

As already mentioned, we view the axioms A_{DS} as 'structural rules' and the equations E_{DS} as 'simplification rules'. Note that the axioms in A_{DS} allow us to write expressions in a simplified manner by dropping unnecessary parentheses in expressions and by making immaterial the order of the arguments. For example, the terms $p_9 \equiv ((p_9 \vee p_2) \equiv p_6)$ can equivalently be written as $p_6 \equiv p_9 \equiv (p_2 \vee p_9)$. Furthermore, adopting our binding convention, the latter expression can be written as $p_6 \equiv p_9 \equiv p_2 \vee p_9$.

We want to use T_{DS} as an admissible simplification system modulo A_{DS}. In that case, T_{DS} can be employed as *rewrite system* $\rightarrow_{E_{\mathrm{DS}}/A_{\mathrm{DS}}}$ on the equivalence classes of $T_{\mathrm{DS}}/A_{\mathrm{DS}}$, or more generally, on $T_{\Sigma(X)_{\mathrm{DS}}/A_{\mathrm{DS}}}$, where X denotes a set of propositional variables added as extra constants to Σ_{DS}.

Theorem 1. *The equations E_{DS} in T_{DS} are confluent and terminating modulo A_{DS}.*

Proof. Termination and confluence modulo A can be established mechanically by using formal tools that: (i) find a well-founded ordering

$\succ$ on A-equivalence classes of terms such that $[t]_A \to_{E/A} [t']_A$ implies $[t]_A \succ [t']_A$, and (ii) check confluence of E modulo A by computing all so-called 'critical-pairs' modulo A and showing they are all confluent. We have used the CiME tool [19] to mechanically check termination and confluence of E_{DS} modulo A_{DS}.

Lemma 1. *The* canonical form *of any Boolean proposition is either* T*,* F *or* $t_0 \equiv \ldots \equiv t_n$*, for some* $n \geq 0$*, where all* t_i *are distinct disjunctions (modulo AC) of propositional variables.*

Proof. We have given a mechanical proof of this fact using Maude's Sufficient Completeness Checker [15]. This can be found in [25]. Intuitively, if the canonical form does not follow such pattern it is easy to check that at least one more equation in E_{DS} could be used to simplify it, thus contradicting the fact that it is canonical.

Since the T_{DS} is both confluent and terminating, and since, as shown in [25], T_{DS} is an equational theory *isomorphic* to the standard Boolean theory, we can use T_{DS} as a *decision procedure for propositional logic.* Moreover, since both T and F are canonical forms, this system has very convenient properties. That is, we have the following equivalences for any propositional expressions t and t':

$$T_{\text{DS}} \vdash t = t' \;\Leftrightarrow\; T_{\text{DS}} \vdash t \equiv t' = \mathsf{T} \;\Leftrightarrow\; \mathsf{can}_{E_{\text{DS}}/A_{\text{DS}}}[t] = \mathsf{can}_{E_{\text{DS}}/A_{\text{DS}}}[t']\,.$$

In particular, since T and F are both in $E_{\text{DS}}/A_{\text{DS}}$-canonical form (or simply, DS-canonical form), we have:

$$T_{\text{DS}} \vdash t \equiv t' = \mathsf{T} \;\Leftrightarrow\; \mathsf{can}_{E_{\text{DS}}/A_{\text{DS}}}[t \equiv t'] = [\mathsf{T}]$$

and

$$T_{\text{DS}} \vdash t \equiv t' = \mathsf{F} \;\Leftrightarrow\; \mathsf{can}_{E_{\text{DS}}/A_{\text{DS}}}[t \equiv t'] = [\mathsf{F}]\,.$$

We call a proposition t a *tautology* iff $\mathsf{can}_{E_{\text{DS}}/A_{\text{DS}}}[t] = [\mathsf{T}]$ and a *falsity* iff $\mathsf{can}_{E_{\text{DS}}/A_{\text{DS}}}[t] = [\mathsf{F}]$. We call t *satisfiable* iff $\mathsf{can}_{E_{\text{DS}}/A_{\text{DS}}}[t] \neq [\mathsf{F}]$. Therefore, our decision procedure gives also a decision procedure for checking satisfiability of any proposition t.

At this point, we note an interesting and useful fact about sequences of equivalences. An *interpretation* of a Boolean proposition t is determined by an assignment or function $\sigma : \{p_0, \ldots, p_n\} \to \{\mathsf{t}, \mathsf{f}\}$ from the set of propositional variables $\text{Vars}(t) = \{p_0, p_1, \ldots, p_n\}$ occurring in t to the set $\{\mathsf{t}, \mathsf{f}\}$ in the following way.

Definition 2. *Given an assignment* $\sigma : \mathrm{Vars}(t) \to \{\mathsf{t}, \mathsf{f}\}$*, the* interpretation $\sigma\langle t\rangle$ *of a Boolean proposition* t *under the assignment* σ *is defined inductively by:*

- $\sigma\langle \mathsf{T}\rangle = \mathsf{t}$ *,*
- $\sigma\langle \mathsf{F}\rangle = \mathsf{f}$ *,*
- $\sigma\langle p_i\rangle = \sigma(p_i)$ *, for propositional variable* p_i*, where* $\sigma(p_i)$ *denotes the value of* p_i *under* σ*,*
- $\sigma\langle \neg t\rangle = \neg\sigma\langle t\rangle$ *, and*
- $\sigma\langle t_1 \circ t_2\rangle = \sigma\langle t_1\rangle \circ \sigma\langle t_2\rangle$ *, for any other operation* $\circ \in \Sigma^{(2)}_{\mathrm{DS}}$*,*

where all operations in Σ_{DS} *are interpreted in* $\{\mathsf{t}, \mathsf{f}\}$ *in the usual way.*

Lemma 2. *Let* $t = t_0 \equiv t_1 \equiv \cdots \equiv t_n$*, for* $n \geq 0$ *and let* σ *be any interpretation.* $\sigma\langle t\rangle = \mathsf{t}$ *iff* $\sigma\langle t_i\rangle = \mathsf{f}$ *for an even number of* t_i*,* $0 \leq i \leq n$*.*

Proof. By induction on n.

Recall that the *DS-canonical form* of any Boolean proposition t is either T, F or $t_0 \equiv \cdots \equiv t_n$, for some $n \geq 0$, where all t_i are distinct disjunctions (modulo AC) of propositional variables. We now show how one can easily 'extract/build' two particular assignments from DS-canonical forms. We say that a DS-canonical form is *sorted* if it is sorted in ascending order by a lexicographic order on the number of propositional variables occurring in each disjunction and by the corresponding multi-set ordering over the indexes of the propositional variables occurring in each disjunction, if the number of variables coincides. Since all the disjuncts t_i are different, this partial order is indeed a total order.

Problem 1. Build an assignment σ from the sorted DS-canonical form $t_0 \equiv \cdots \equiv t_n$ such that $\sigma\langle t_0 \equiv \cdots \equiv t_n\rangle = \mathsf{t}$.

Solution We give a simple solution by cases that maximizes the number of true assignments in σ :

- $n = 0$ and $t_0 = \mathsf{T}$. This is the case of a tautology. Therefore, any assignment will satisfy the expression. Then, our σ is such that $\sigma(p_i) = \mathsf{t}$ for all i.
- $n = 0$ and $t_0 = \mathsf{F}$. This is the case of a falsity, and therefore, there is no assignment σ satisfying the expression.
- $n = 0$ and $t_0 \neq \mathsf{T}$ and $t_0 \neq \mathsf{F}$. Similar to the case $n = 0$ and $t_0 = \mathsf{T}$.

- $n \geq 1$ and $t_0 \neq \mathsf{F}$. This is the case where there are at least two disjunctions in the sequence of equivalences and F does not occur. Therefore, by Lemma 2, σ assigning all propositional variables to t entails that σ satisfies the sequence of equivalences. Therefore, σ is the assignment where $\sigma(p_i) = \mathsf{t}$ for all i.
- $n \geq 1$ and $t_0 = \mathsf{F}$. This is the case where there are at least two disjunctions in the sequence of equivalences and the first of them corresponds to the empty one, that is, to F. Since in any canonical form all disjunctions are different, there is only one occurrence of F in the sequence of equivalences. By Lemma 2, and forcing $\sigma\langle t_1 \rangle = \mathsf{f}$ and $\sigma\langle t_j \rangle = \mathsf{t}$ $(2 \leq j \leq n)$, we alter the parity of f in the sequence of equivalences, and minimize the assignments to f, while requiring σ to satisfy the sequence of equivalences. Then, σ is the assignment satisfying $\sigma(p_i) = \mathsf{t}$ if $p_i \notin \mathrm{Vars}(t_1)$ and $\sigma(p_i) = \mathsf{f}$ if $p_i \in \mathrm{Vars}(t_1)$. □

Problem 2. Build an assignment σ from the DS-canonical form $t_0 \equiv \cdots \equiv t_n$ such that $\sigma\langle t_0 \equiv \cdots \equiv t_n \rangle = \mathsf{f}$.

Solution. First observe that, since σ is a homomorphic function, $\sigma\langle t \rangle = \mathsf{f}$ iff $\neg\sigma\langle t \rangle = \mathsf{t}$ iff $\sigma\langle \neg t \rangle = \mathsf{t}$, for any propositional expression t. But $\neg(t_0 \equiv \cdots \equiv t_n)$ is not in DS-canonical form. Since $\neg t = t \equiv \mathsf{F}$, $\neg(t_0 \equiv \cdots \equiv t_n)$ is logically equivalent to $t_0 \equiv \cdots \equiv t_n \equiv \mathsf{F}$, which may still not be in DS-canonical form. We then have the following cases in order to have a sorted DS-canonical form t' semantically equivalent to this last sequence of equivalences:

- $n = 0$ and $t_0 = \mathsf{T}$. Clearly $t' = \mathsf{F}$.
- $n = 0$ and $t_0 = \mathsf{F}$. Clearly $t' = \mathsf{T}$.
- $n \geq 1$ and $t_0 = \mathsf{F}$. Then $t' = t_1 \equiv \cdots \equiv t_n$.
- $n \geq 1$ and $t_0 \neq \mathsf{F}$. Then $t' = \mathsf{F} \equiv t_0 \equiv \cdots \equiv t_n$.

In either case it is easy to check that t and $\neg t'$ are logically equivalent. Thus, in order to obtain σ, apply to t' the solution to Problem 1. The resulting assignment σ will satisfy $\sigma\langle t \rangle = \mathsf{f}$. □

Assuming that the canonical form $t = t_0 \equiv \cdots \equiv t_n$ is a sorted sequence of equivalences in which there occur at most m propositional variables, we have that both problems are solvable in $O(m)$; for all the cases, except when $n \geq 1$ and $t_0 = \mathsf{F}$, the problems are solvable in $O(1)$. Furthermore, both solutions provide an assignment that maximizes the number of t valuations on the propositional variables.

Decision procedures for propositional logic have been a matter of interest in the rewriting community for a long time. The first decision procedure for propositional logic based on rewriting was developed by J. Hsiang in the early 1980s [16]. His result follows from a previously known connection between *Boolean algebras* and *Boolean rings* due to M.H. Stone: every Boolean algebra, if properly viewed, is a special type of ring called a *Boolean ring* [17, 26]. As a matter of fact, T_{DS} is *dual* to Hsiang's canonical system [25]. The close connections between the work of Dijkstra-Scholten and that of Hsiang seem to have passed unnoticed until now.

Various first-order and modal propositional theorem proving systems have used canonical systems for propositional logic. A refutation rewrite-based method, the *N-strategy*, was designed by Hsiang for the word problem in first-order predicate logic in 1985 [18]. The N-strategy used the canonical system for Boolean algebras to reduce a set of clauses to canonical form and then converting the clauses into a set of simplifying equations. Following a similar approach, L. Bachmair and N. Dershowitz developed a complete inference system for first-order theorem proving based on Hsiang's rewrite system for Boolean algebras [1]. D. Kapur and P. Narendran in 1985, following a different approach, developed another refutation method for first-order theorem proving. In 1992 A. Foret extended Hsiang's system to give a complete term rewriting system for the modal propositional logics known as K, Q, T and S5 [10]. In the same spirit, we will extend the DS-decision procedure to a decision procedure for Syllogistic Logic with Complements and Boolean connectives in Section 5.

4 Specification in Maude

In this section we present the specification in Maude of the equational theory T_{DS}. Maude is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming for a wide range of applications. Maude functional modules are equational theories. We define the module `BOOL-DS` to represent the theory T_{DS} in the following way:

```
fmod BOOL-DS is
   ...
endfm
```

where the contents expressed in ... is explained in detail in what follows. The first thing a specification needs to declare are the types (that in the algebraic specification community are usually called *sorts*) of the data being defined and the corresponding operations. A sort is declared using the `sort` keyword followed by an identifier (the sort name), followed by white space and a period. In our Maude specification of T_{DS} we use the sort `BoolDS` as the main sort of the specification. In this way, any term we compute with in our specification has sort `BoolDS`. We specify this in Maude as follows:

```
sort BoolDS .
```

In a Maude module, an operator is declared with the keyword `op` (alternatively `ops` if more than one operation is defined simultaneously) followed by its name (or names separated with white space), followed by a colon, followed by the list of sorts for its arguments (called the operator's arity or domain sorts), followed by `->`, followed by the sort of its result (called the operator's coarity or range sort), optionally followed by an attribute declaration, followed by white space and a period. We show various operator declarations in the following paragraphs.

First, we model the constants `F` and `T` as `FALSE` and `TRUE`, respectively.

```
ops FALSE TRUE : -> BoolDS .
```

We use Maude's built-in sort `Nat` to generate propositional variables. This is accomplished by importing the module `NAT` (the one defining the sort `Nat`) and declaring an operation `p`:

```
protecting NAT .
op p : Nat -> BoolDS .
```

The specification of the unary and binary function symbols in Σ_{DS}, with their binding power for parsing purposes, is specified in Maude as follows:

```
op not_ : BoolDS -> BoolDS [prec 10] .
op _equ_ : BoolDS BoolDS -> BoolDS [assoc comm prec 80] .
op _neq_ : BoolDS BoolDS -> BoolDS [assoc comm prec 80] .
op _or_  : BoolDS BoolDS -> BoolDS [assoc comm prec 50] .
op _and_ : BoolDS BoolDS -> BoolDS [assoc comm prec 50] .
op _imp_ : BoolDS BoolDS -> BoolDS [prec 70] .
op _cos_ : BoolDS BoolDS -> BoolDS [prec 70] .
```

We have used `not` for $\neg$, `equ` for $\equiv$, `neq` for $\not\equiv$, `or` for $\vee$, `and` for $\wedge$, `imp` for $\Rightarrow$ and `cos` for $\Leftarrow$. In Maude, the user can specify each operator with its own syntax, which can be prefix, postfix, infix, or any 'mixfix' combination. This is done by indicating with underscores the places where the arguments appear in the mixfix syntax. For example, `not` is a prefix unary operator while `and` is an infix binary operator. In practice, this improves readability (and therefore understandability) of programs and data. The *attributes* of an operation are specified within square brackets. In our case, we use `assoc` (associativity) and `comm` (commutativity) for *equational* attributes that can be used to reason modulo such axioms, and `prec` (precedence–the higher the value, the lower the binding power) for the *parsing precedence* attribute. Note that, in this case, the equational attributes of the operators correspond to the equations in A_{DS} and the `prec` attribute specifies the syntax's binding policy we defined previously.

Equations are declared using the keyword `eq`, followed by a term (its lefthand side), the equality sign `=`, then a term (its righthand side), followed by white space and a period. The set E_{DS} of equations is specified as follows:

```
vars P Q R : BoolDS .
eq P equ TRUE = P .
eq P equ P = TRUE .
eq not P = P equ FALSE .
eq P neq Q = P equ Q equ FALSE .
eq P or P = P .
eq P or ( Q equ R ) = P or Q equ P or R .
eq P or TRUE = TRUE .
eq P or FALSE = P .
eq P and Q = P equ Q equ P or Q .
eq P imp Q = P or Q equ Q .
eq P cos Q = Q imp P .
```

where `P`, `Q` and `R` are variables of sort `BoolDS`. In this way, the theory T_{DS} has been specified in Maude as an executable specification, and we can then use it to simplify terms. That is, since Maude efficiently implements the equational simplification relation $\rightarrow_{E/A}$, and the equations E_{DS} are confluent and terminating modulo A_{DS}, the Maude specification of T_{DS} is automatically a decision procedure for propositional logic.

As an example, we use this specification to solve the following instance of the word problem, involving two expressions commonly appearing in the specification of software postconditions. Are the propositions $(p_0 \wedge p_1) \vee (\neg p_0 \wedge p_2)$ and $(p_0 \Rightarrow p_1) \wedge (\neg p_0 \Rightarrow p_2)$ equal?

One way of answering the question is by reducing both expressions to their canonical form. If the canonical forms coincide, then the terms are semantically equal. Indeed, it is the case that both canonical forms coincide, since both are:

$$p_1 \equiv p_0 \vee p_1 \equiv p_0 \vee p_2 \, .$$

In Maude, we use the `red` (reduce) command to simplify the corresponding expressions to canonical forms. The result is shown below:

```
Maude> red (p(0) and p(1)) or (not p(0) and p(2)) .
reduce in BOOL-DS : (p(0) and p(1)) or p(2) and
       not p(0) .
rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
result BoolDS: p(1) equ p(0) or p(1) equ p(0) or p(2)

Maude> red (p(0) imp p(1)) and (not p(0) imp p(2)) .
reduce in BOOL-DS : (p(0) imp p(1)) and
       (not p(0) imp p(2)) .
rewrites: 12 in 0ms cpu (0ms real) (~ rewrites/second)
result BoolDS: p(1) equ p(0) or p(1) equ p(0) or p(2)
```

An equivalent way of solving the word problem, is by using the $\equiv$ symbol as the Boolean equality. By computing the canonical form of the expression:

$$(p_0 \wedge p_1) \vee (\neg p_0 \wedge p_2) \;\equiv\; (p_0 \Rightarrow p_1) \wedge (\neg p_0 \Rightarrow p_2)\,,$$

one can corroborate that T is the canonical form of the previous, tautological expression.

The corresponding simplification in Maude is the following:

```
Maude> red (p(0) and p(1)) or (not p(0) and p(2)) equ
       (p(0) imp p(1)) and (not p(0) imp p(2)) .
reduce in BOOL-DS : (p(0) and p(1)) or p(2) and not p(0)
       equ (p(0) imp p(1)) and (not p(0) imp p(2)) .
rewrites: 40 in 0ms cpu (0ms real) (~ rewrites/second)
result BoolDS: TRUE
```

Either way we have found that $(p_0 \wedge p_1) \vee (\neg p_0 \wedge p_2)$ and $(p_0 \Rightarrow p_1) \wedge (\neg p_0 \Rightarrow p_2)$ are semantically equivalent propositions.

In terms of performance, a very exhaustive comparison of this executable specification in Maude with another three decision procedures for propositional logic can be found in [25]. On average, T_{DS} behaves among the best ones.

5 A Decision procedure for Syllogistic Logic with Complements

We have been investigating the use of equational theories in various domains as tools of automated deduction. Our motivation is to find simpler methods in which replacement of equals by equals is powerful enough to solve the corresponding decision problem.

Syllogistic Logic, the logic of Aristotle, was widely considered to be the universal foundation for all deductive reasoning for many centuries. According to Aristotle, 'a syllogism is a discourse in which, certain things having been supposed, something different from the things supposed results of necessity because these things are so.' A *syllogism* consists of three syllogistic propositions, two premises (or hypotheses) and one conclusion (or thesis). More precisely, a *syllogistic proposition* is a sentence asserting or denying a fact of the form $\mathbf{A}PQ$ (All P are Q), $\mathbf{E}PQ$ (No P is Q), $\mathbf{I}PQ$ (Some P is Q) and $\mathbf{O}PQ$ (Some P are not Q), where P and Q are monadic predicates representing subsets of some non-empty universe of objects. A syllogism is considered *valid* if its thesis follows from its two hypotheses by applying the Aristotelian deduction rules which, from the logical point of view, are particular cases of those of First-Order Logic. For example, 'All men are mortal and all Colombians are men, then all Colombians are mortal' is a valid syllogism. It was G. Frege who pointed out the need for a more general logical system in order to express the foundations of all deductive reasoning. Since Frege, FOL has been the main subject of study for logicians interested in this area. In the past few years, the interest in Syllogistic Logic has experienced an special awakening since 'specialists in communication and information theory employ ideas which can be traced back to Aristotle's work' [11].

Traditionally, Syllogistic Logic has been studied by philosophers and logicians pursuing different interests. While philosophers strictly adhere to the initial ideas of Aristotle and are users of the logical system, modern logicians have been interested in studying the symbolic properties of the system itself[2]. In the recent history of symbolic logic, it was P. Bernays, in Hilbert's Göttingen school, who first introduced the use of propositional Boolean connectives and FOL quantifiers to capture all the logical relationships dealt with in Aristotelian syllogistic logic. It was J. Łukasiewicz who proposed a sound and complete axiomatization of Syllogistic Logic within the logical framework of First-Order Logic [21]. Moreover, he showed that Syllogistic Logic treated from the logicians point of view is a proper fragment of Monadic-First Order Logic (MFOL). MFOL is the fragment of First Order Logic in which predicates are monadic, i.e., have exactly one argument. Since MFOL is decidable [3], Syllogistic Logic is too.

The standard representation of syllogistic logic in FOL is straightforward, and it is summarized in Table 1. In a sense, Moss' representation

[2] We refer the reader to [8, 11, 21] for detailed discussions of these two different sets of ideas.

of syllogisms in CSYLL is simpler, since it does not require an explicit treatment of quantifiers as in the standard FOL representation.

Table 1. Standard representation of syllogistic propositions in FOL.

Syllogistic prop.	FOL formula
$\mathbf{A}PQ$	$(\forall x)(P(x) \Rightarrow Q(x))$
$\mathbf{E}PQ$	$(\forall x)(P(x) \Rightarrow \neg Q(x))$
$\mathbf{I}PQ$	$(\exists x)(P(x) \wedge Q(x))$
$\mathbf{O}PQ$	$(\exists x)(P(x) \wedge \neg Q(x))$

First, we present the language $\mathcal{L}$ and deductive system $K_{\mathcal{L}}$ of L.S. Moss using the Boolean algebra connectives of Section 3. Secondly, we propose an equational theory T_{CSYLL} for the general case of CSYLL, using the aforementioned Boolean connectives, and we prove that it provides a sound and complete (with respect to $K_{\mathcal{L}}$) decision procedure for the Syllogistic Logic with Complements and Boolean Connectives. Finally, we present three examples in which an executable specification of T_{CSYLL} in Maude is used with sample syllogisms.

5.1 Syllogistic Logic with Complement and Boolean Connectives

L.S. Moss has recently proposed an axiomatization of the Syllogistic Logic with Complements (CSYLL) in which some Boolean connectives and the complement relation are allowed in syllogisms, thus obtaining a more general, yet complete and sound, logic than the traditional syllogistic one [23]. We will, furthermore, allow all the Boolean connectives of T_{DS} in CSYLL. Our main goal, then, will be to have a rewriting-based *decision procedure* for CSYLL by extending the decision procedure for propositional logic presented in Sections 3 and 4. Then, we will not be only interested in 'classic syllogisms' but in their 'modern versions' in which the Boolean connectives can be freely used. In this subsection, we define the original system proposed by Moss and mention as main result its soundness and completeness.

We use the set Π of monadic predicates (predicates for short) $P, Q, \ldots$, which in turn represent plural common nouns, to parameterize the language of Syllogistic Logic with Complement and Boolean Connectives.

Definition 3. *A* Π-model $\mathcal{M} = (M, \_\mathcal{M})$ *is a non-empty set* M *and an interpretation function* $\_\mathcal{M} : \Pi \to \mathcal{P}(M)$ *interpreting each monadic predicate* P *as a subset* $P_\mathcal{M} \subseteq M$ *(allowing* $P_\mathcal{M}$ *to be empty).*

We take the syntax and semantics of this language, as defined in [23], which we will call $\mathcal{L}(\Pi)$. We consider *sentences* 'All P are Q' and 'Some P are Q', for $P, Q \in \Pi$, which are commonly denoted with the letter S. We assume that there is a *complementation operation* $\_^C$ over the predicates such that $(P^C)^C = P$, for any $P \in \Pi$. Furthermore, sentences can be operated on with the classical propositional operators as in '$\neg$(Some P are Q^C) $\wedge$ (All Q are R)'. Let us use the following EBNF to precisely define the language $\mathcal{L}(\Pi)$:

Definition 4. *We define* $\mathcal{L}(\Pi)$*, for any* $P \in \Pi$ *and* Atoms A *and* B*, as follows:*

$$\begin{aligned} &\text{Atom} \quad ::= P \mid P^C \\ &\text{Sentence} ::= \textit{All } A \textit{ are } B \;\mid\; \textit{Some } A \textit{ are } B \;\mid \neg(\text{Sentence }) \mid (\text{Sentence}) \circ (\text{Sentence}) \ , \end{aligned}$$

where $\circ$ *stands for any binary operator in* Σ_{DS}.

Definition 5. *We say that a* Π*-model* $\mathcal{M}$ satisfies *a* $\mathcal{L}(\Pi)$*-sentence* S*, written* $\mathcal{M} \models S$*, according to the following equivalences:*

$$\begin{aligned} &\mathcal{M} \models (\textit{All } A \textit{ are } B) \quad \textit{iff } A_\mathcal{M} \subseteq B_\mathcal{M} \ , \\ &\mathcal{M} \models (\textit{Some } A \textit{ are } B) \ \textit{iff } A_\mathcal{M} \cap B_\mathcal{M} \neq \emptyset \ , \end{aligned}$$

where we define $(A^C)_\mathcal{M} = M - A_\mathcal{M}$ *and we treat the Boolean connectives classically.*

For a Π model $\mathcal{M}$, if Γ is a set of $\mathcal{L}(\Pi)$-sentences, then we write $\mathcal{M} \models \Gamma$ to denote $\mathcal{M} \models S$ for every $S \in \Gamma$, and $\Gamma \models S$ to denote that every model satisfying every sentence in Γ also satisfies S.

Definition 6. *Let* A*,* B *and* C *be* $\mathcal{L}(\Pi)$*-atoms and* S *a* $\mathcal{L}(\Pi)$*-sentence. The* system $K_\mathcal{L}$ *is a Hilbert-style one, having* modus ponens *as the only inference rule, and with the following axioms:*

1. *All substitution instances of propositional tautologies*
2. *All* A *are* A
3. $(\textit{All } A \textit{ are } C) \wedge (\textit{All } C \textit{ are } B) \Rightarrow (\textit{All } A \textit{ are } B)$
4. $(\textit{All } B \textit{ are } C) \wedge (\textit{Some } A \textit{ are } B) \Rightarrow (\textit{Some } C \textit{ are } A)$

5. $(\mathit{Some\ A\ are\ B}) \Rightarrow\ (\mathit{Some\ A\ are\ A})$
6. $\neg(\mathit{Some\ A\ are\ A}) \Rightarrow\ (\mathit{All\ A\ are\ B})$
7. $(\mathit{Some\ A\ are\ B^C}) \equiv \neg(\mathit{All\ A\ are\ B})$ *.*

We call S a theorem *of $K_{\mathcal{L}}$, written $\vdash_{\mathcal{L}} S$, iff there is a* proof *of S in $K_{\mathcal{L}}$, that is, a sequence of $\mathcal{L}(\Pi)$-sentences $\psi_0, \psi_1, \ldots, \psi_n$ such that $\psi_n = S$ and for each ψ_i, $0 \leq i \leq n$, either ψ_i is an axiom of $K_{\mathcal{L}}$ or ψ_i follows from two previous members of the sequence, say ψ_j and ψ_k ($0 \leq j < k < i$), as a direct consequence of using modus ponens.*

Given a set Γ of $\mathcal{L}(\Pi)$-sentences and an $\mathcal{L}(\Pi)$-sentence S, we define the relation $\Gamma \vdash_{\mathcal{L}} S$ by the equivalence:

$$\Gamma \vdash_{\mathcal{L}} S \quad \text{iff} \quad \vdash_{\mathcal{L}} (S_0 \wedge \cdots \wedge S_n) \Rightarrow S\,,$$

for some $S_0, \ldots, S_n \in \Gamma$.

We now present the soundness and completeness theorem for $K_{\mathcal{L}}$ proven in [23].

Theorem 2. *Given a set Γ of $\mathcal{L}$-sentences and an $\mathcal{L}$-sentence S,*

$$\Gamma \models S \quad \text{iff} \quad \Gamma \vdash_{\mathcal{L}} S\,,$$

i.e., $K_{\mathcal{L}}$ is sound and complete.

Notice that, since $K_{\mathcal{L}}$ is an axiomatization at least as 'strong' as that of Łukasiewicz, any decision procedure for $K_{\mathcal{L}}$ is a decision procedure for the axiomatization of Łukasiewicz, and therefore for the traditional syllogistic system.

5.2 The equational theory T_{CSYLL}

We present the theory $T_{\text{CSYLL}} = (\Sigma_{\text{CSYLL}}, E_{\text{CSYLL}} \uplus A_{\text{CSYLL}})$ as a many-sorted equational theory with sorts *Term* and *Sentence*.

Definition 7. *The theory $T_{\text{CSYLL}} = (\Sigma_{\text{CSYLL}}, E_{\text{CSYLL}} \uplus A_{\text{CSYLL}})$ is defined as follows. Σ_{CSYLL} has two sorts,* Term *and* Sentence*, with the following function symbols:*

$$\begin{array}{ll}
\mathsf{T}, \mathsf{F} & : \to \mathit{Term} \\
\neg & : \mathit{Term} \to \mathit{Term} \\
\equiv, \not\equiv, \vee, \wedge, \Rightarrow, \Leftarrow & : \mathit{Term}\ \mathit{Term} \to \mathit{Term} \\
\mathsf{T}, \mathsf{F} & : \to \mathit{Sentence} \\
\neg & : \mathit{Sentence} \to \mathit{Sentence} \\
\equiv, \not\equiv, \vee, \wedge, \Rightarrow, \Leftarrow & : \mathit{Sentence}\ \mathit{Sentence} \to \mathit{Sentence} \\
[\_], \{\_\} & : \mathit{Term} \to \mathit{Sentence}\ .
\end{array}$$

The equations A_{CSYLL} correspond to the equations in A_{DS} for both sorts Term *and* Sentence*. That is, if we use $A_{\text{DS}}^{\text{Term}}$ and $A_{\text{DS}}^{\text{Sentence}}$ to denote the set A_{DS} over the sort* Term *and* Sentence*, respectively, we have:*

$$A_{\text{CSYLL}} = A_{\text{DS}}^{\text{Term}} \cup A_{\text{DS}}^{\text{Sentence}} \ .$$

Accordingly, if we denote with $E_{\text{DS}}^{\text{Term}}$ and $E_{\text{DS}}^{\text{Sentence}}$ the two extensions of E_{DS} over the sort Term *and* Sentence*, respectively, we have for P, Q : Term:*

$$\begin{aligned} E_{\text{CSYLL}} = E_{\text{DS}}^{\text{Term}} \cup E_{\text{DS}}^{\text{Sentence}} \cup \{\, & [P] = \neg\{\neg P\}, \\ & \{\mathsf{T}\} = \mathsf{T}, \{\mathsf{F}\} = \mathsf{F}, \{P\} \vee \{Q\} = \{P \vee Q\} \,\}. \end{aligned}$$

Definition 8. *A* model *of T_{CSYLL} is a Σ_{CSYLL}-algebra that satisfies the equations $E_{\text{CSYLL}} \cup A_{\text{CSYLL}}$. Let M be a non-empty set. We can define a Σ_{CSYLL}-algebra $\mathcal{P}(M)$ interpreted in the sort* Term *by the power set $\mathcal{P}(M)$ and with its operations interpreted as the usual operations in the algebra of sets; the sort* Sentence *is interpreted by the 2-element set $\{\mathsf{t}, \mathsf{f}\}$, with the Boolean operators interpreted in the standard way as well. Given $X \subseteq M$ we define:*

$$\begin{aligned} [X] \ &= \text{if } X = M \text{ then } \mathsf{t} \text{ else } \mathsf{f} \ , \\ \{X\} &= \text{if } X \neq \emptyset \text{ then } \mathsf{t} \text{ else } \mathsf{f} \ . \end{aligned}$$

Proposition 1. *For a non-empty set M, $\mathcal{P}(M)$ satisfies the equations in $E_{\text{CSYLL}} \cup A_{\text{CSYLL}}$.*

Proof. Since T_{DS} is isomorphic to the traditional Boolean theory [25], the equations in $A_{\text{DS}}^{\textit{Term}}$, $A_{\text{DS}}^{\textit{Sentence}}$, $E_{\text{DS}}^{\textit{Term}}$ and $E_{\text{DS}}^{\textit{Sentence}}$ are satisfied by $\mathcal{P}(M)$. We prove that $\mathcal{P}(M)$ satisfies the equation $[P] = \neg\{\neg P\}$. Let $X \subseteq M$. If $[X] = \mathsf{t}$, then $X = M$ and therefore $\neg X = \emptyset$. It follows that $\{\neg X\} = \mathsf{f}$ implying $\neg\{\neg X\} = \mathsf{t}$. Conversely, if $[X] = \mathsf{f}$, we have two cases: if $X = \emptyset$, then $\neg X = M$ and therefore $\neg\{\neg X\} = \mathsf{f}$; if $X \neq \emptyset$, and since $X \neq M$, then $\{\neg X\} = \mathsf{t}$ and $\neg\{\neg X\} = \mathsf{f}$. The satisfaction of $\mathcal{P}(M)$ for the remaining equations is proved similarly.

Notice that a Π-model $\mathcal{M}$ determines a unique T_{CSYLL}-homomorphism $\_{\mathcal{M}} : T_{\Sigma_{\text{CSYLL}}}(\Pi) \to \mathcal{P}(M)$ such that the following diagram commutes:

$$\begin{tikzcd}
\Pi \arrow[r, hook] \arrow[dr, "\_{\mathcal{M}}"'] & T_{\Sigma_{\mathrm{CSYLL}}}(\Pi) \arrow[d, dashed, "\_{\mathcal{M}}"] \\
 & \mathcal{P}(M)
\end{tikzcd}$$

where $T_{\Sigma_{\mathrm{CSYLL}}}(\mathit{\Pi})$ is the free Σ_{CSYLL}-algebra on the monadic predicates $\mathit{\Pi}$, that is, the algebra of all Σ_{CSYLL}-terms formed on the variables $\mathit{\Pi}$. In other words, the Σ_{CSYLL}-homomorphism allows us to interpret in the expected way the satisfaction of all Σ_{CSYLL}-sentences as either true or false in $\mathcal{M}$.

Note that $\mathcal{L}$ can also be viewed as a two-sorted signature with sorts *Atom* and *Sentence*, and operations[3]:

$$\begin{array}{ll}
\_^{C} & : \mathit{Atom} \rightarrow \mathit{Atom} \\
(\text{All}\ \_\ \text{are}\ \_) & : \mathit{Atom}\ \mathit{Atom} \rightarrow \mathit{Sentence} \\
(\text{Some}\ \_\ \text{are}\ \_) & : \mathit{Atom}\ \mathit{Atom} \rightarrow \mathit{Sentence} \\
\mathsf{T}, \mathsf{F} & : \rightarrow \mathit{Sentence} \\
\neg & : \mathit{Sentence} \rightarrow \mathit{Sentence} \\
\equiv, \not\equiv, \vee, \wedge, \Rightarrow, \Leftarrow & : \mathit{Sentence}\ \mathit{Sentence} \rightarrow \mathit{Sentence}\ .
\end{array}$$

Definition 9. *We define the obvious signature morphism $H : \mathcal{L} \rightarrow \Sigma_{\mathrm{CSYLL}}$, mapping the sorts* Atom *to* Term *and* Sentence *to* Sentence*, $(\ )^{C}$ to $\neg : Term \rightarrow Term$, (All A are B) to $[A \Rightarrow B]$ and (Some A are B) to $\{A \wedge B\}$. H maps identically the Boolean connectives over the sentences.*

H defines a forgetful functor $\_|_{H} : \mathbf{Alg}_{\Sigma_{\mathrm{CSYLL}}} \rightarrow \mathbf{Alg}_{\mathcal{L}}$. In particular, given any set $\mathit{\Pi}$ of monadic predicates, this induces a homomorphic translation $H : T_{\mathcal{L}}(\mathit{\Pi}) \rightarrow T_{\Sigma_{\mathrm{CSYLL}}}(\mathit{\Pi})$, namely, the unique $\mathcal{L}$-homomorphism commuting the following diagram:

$$\begin{tikzcd}
\Pi \arrow[r, hook] \arrow[dr, hook] & T_{\mathcal{L}}(\Pi) \arrow[d, dashed, "H"] \\
 & T_{\Sigma_{\mathrm{CSYLL}}}(\Pi)|_{H}
\end{tikzcd}$$

Furthermore, $\mathcal{P}(M)|_{H}$ is an $\mathcal{L}$-algebra, so that we can reinterpret the $\mathcal{L}(\mathit{\Pi})$-semantics in a $\mathit{\Pi}$-model $\mathcal{M}$ as the unique $\mathcal{L}$-homomorphism $\_{}_{\mathcal{M}}$ commuting the diagram:

[3] The motivation for extending $\mathcal{L}$ with the *sentences* T and F will become clear in the following paragraphs.

$$\begin{tikzcd}
\Pi \arrow[r, hook] \arrow[dr, "{\_}_{\mathcal{M}}"'] & T_{\mathcal{L}}(\Pi) \arrow[d, dashed, "{\_}_{\mathcal{M}}"] \\
 & \mathcal{P}(M)|_H
\end{tikzcd}$$

which provides an algebraic semantics for $\mathcal{L}(\varPi)$. We can then, alternatively, view the pairs $(\mathcal{P}(M)|_H, {\_}_{\mathcal{M}})$ and $(\mathcal{P}(M), {\_}_{\mathcal{M}})$ as, respectively, $\mathcal{L}(\varPi)$-algebras and $\varSigma_{\mathrm{CSYLL}}(\varPi)$-algebras, when the monadic predicates $\varPi$ are added as extra constants. Moreover, we have the signature homomorphism $H_{\varPi} : \mathcal{L}(\varPi) \to \varSigma_{\mathrm{CSYLL}}(\varPi)$ extending H and being the identity on $\varPi$, inducing a functor ${\_}|_{H_{\varPi}} : \mathbf{Alg}_{\varSigma_{\mathrm{CSYLL}}(\varPi)} \to \mathbf{Alg}_{\mathcal{L}(\varPi)}$ such that:

$$(\mathcal{P}(M), {\_}_{\mathcal{M}})|_{H_{\varPi}} = (\mathcal{P}(M)|_H, {\_}_{\mathcal{M}}).$$

We say that the $\varSigma_{\mathrm{CSYLL}}$-sentence S is a *theorem* of T_{CSYLL}, written $\vdash_{\mathrm{CSYLL}} S = \mathsf{T}$, iff $T_{\mathrm{CSYLL}} \vdash S = \mathsf{T}$. Given a set of $\varSigma_{\mathrm{CSYLL}}$-sentences $\varGamma$ and a $\varSigma_{\mathrm{CSYLL}}$-sentence S, we define the relation $\varGamma \vdash_{\mathrm{CSYLL}} S = \mathsf{T}$ by the equivalence:

$$\varGamma \vdash_{\mathrm{CSYLL}} S = \mathsf{T} \qquad \text{iff} \qquad \vdash_{\mathrm{CSYLL}} (S_0 \wedge \cdots \wedge S_n) \Rightarrow S = \mathsf{T}\,,$$

for some $S_0, \ldots, S_n \in \varGamma$.

Theorem 3. *(Soundness) Let S be an $\mathcal{L}(\varPi)$-sentence. If $T_{\mathrm{CSYLL}} \vdash H(S) = \mathsf{T}$ then $\vdash_{\mathcal{L}} S$.*

Proof. Let $\mathcal{M}$ be any $\varPi$-model and suppose $T_{\mathrm{CSYLL}} \vdash H(S) = \mathsf{T}$. Then, because equational logic is sound, we know that $(\mathcal{P}(M), {\_}_{\mathcal{M}}) \models H(S) = \mathsf{T}$. By the *satisfaction condition* for the many-sorted equational logic institution [12], we have the equivalence $(\mathcal{P}(M), {\_}_{\mathcal{M}}) \models H(S) = \mathsf{T}$ iff $(\mathcal{P}(M)|_H, {\_}_{\mathcal{M}}) \models S = \mathsf{T}$. Moreover, since $(\mathcal{P}(M)|_H, {\_}_{\mathcal{M}}) \models S = \mathsf{T}$ iff $\mathcal{M} \models S$, we get that $\mathcal{M} \models S$ is true. Because $K_{\mathcal{L}}$ is sound and complete (Theorem 2) we can conclude $\vdash_{\mathcal{L}} S$. Therefore, T_{CSYLL} is sound.

Theorem 4. *(Completeness) T_{CSYLL} is complete with respect to $K_{\mathcal{L}}$, i.e., for $\varGamma$ a set of $\mathcal{L}(\varPi)$-sentences, and S an $\mathcal{L}(\varPi)$-sentence,*

$$\text{if} \qquad \varGamma \vdash_{\mathcal{L}} S \quad \text{then} \quad H(\varGamma) \vdash_{\mathrm{CSYLL}} H(S) = \mathsf{T}\,,$$

where $H(\varGamma) = \{H(S') \mid S' \in \varGamma\}$.

Proof. By induction on the length of the proof of S in $K_{\mathcal{L}}$. Without loss of generality, let $\psi_1, \ldots, \psi_m$, where $\psi_m = S$, be a minimal length proof for S in $K_{\mathcal{L}}$.

- Case $m = 0$. Necessarily S is either an axiom of $K_{\mathcal{L}}$ or an $\mathcal{L}(\Pi)$-sentence in Γ. In both cases, it is easy to check that $H(S) = \mathsf{T}$, and therefore, $H(\Gamma) \vdash_{\text{CSYLL}} S(H) = \mathsf{T}$. Observe that axiom 1. of $K_{\mathcal{L}}$ follows trivially because T_{CSYLL} extends T_{DS} which is isomorphic to the traditional Boolean theory and therefore complete for propositional logic; theorems 2.-7. have been mechanically proved (see Example 2 at the end of this Section). For the other case, observe that $p_0 \wedge p_1 \Rightarrow p_0$ is a tautology.
- Case $m > 0$. If S is either an axiom of $K_{\mathcal{L}}$ or a sentence in Γ, the proof is similar to that of case $m = 0$. Otherwise, S is obtained by modus ponens from ψ_i and $\psi_j = \psi_i \Rightarrow S$, for $0 \leq i \neq j < m$. Then by the induction hypothesis (IH) we have that $H(\Gamma) \vdash_{\text{CSYLL}} H(\psi_i) = \mathsf{T}$ and $H(\Gamma) \vdash_{\text{CSYLL}} H(\psi_i) \Rightarrow H(S) = \mathsf{T}$. Therefore,

$$
\begin{array}{ll}
 & H(\Gamma) \vdash_{\text{CSYLL}} H(\psi_i) \Rightarrow H(S) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ Definition of } \Gamma \vdash_{\text{CSYLL}} S = \mathsf{T} \rangle \\
 & \vdash_{\text{CSYLL}} H(S_0) \wedge \ldots \wedge H(S_n) \;\Rightarrow\; (H(\psi_i) \Rightarrow H(S)) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ Tautology } \rangle \\
 & \vdash_{\text{CSYLL}} \mathsf{T} \wedge (H(S_0) \wedge \ldots \wedge H(S_n) \;\Rightarrow\; (H(\psi_i) \Rightarrow H(S))) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ IH: } H(\Gamma) \vdash_{\text{CSYLL}} H(\psi) = \mathsf{T} \rangle \\
 & \vdash_{\text{CSYLL}} (H(S_0) \wedge \ldots \wedge H(S_n) \;\Rightarrow\; H(\psi_i)) \wedge (H(S_0) \wedge \ldots \wedge H(S_n) \Rightarrow (H(\psi_i) \Rightarrow H(S))) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ Tautology } \rangle \\
 & \vdash_{\text{CSYLL}} H(S_0) \wedge \ldots \wedge H(S_n) \;\Rightarrow\; (H(\psi_i) \wedge (H(\psi_i) \Rightarrow H(S))) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ Tautology } \rangle \\
 & \vdash_{\text{CSYLL}} H(S_0) \wedge \ldots \wedge H(S_n) \;\Rightarrow\; H(\psi_i) \wedge H(S) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ IH: } H(\Gamma) \vdash_{\text{CSYLL}} H(\psi) = \mathsf{T} \rangle \\
 & \vdash_{\text{CSYLL}} H(S_0) \wedge \ldots \wedge H(S_n) \;\Rightarrow\; \mathsf{T} \wedge H(S) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ Tautology } \rangle \\
 & \vdash_{\text{CSYLL}} H(S_0) \wedge \ldots \wedge H(S_n) \;\Rightarrow\; H(S) = \mathsf{T} \\
\Leftrightarrow & \langle \text{ Definition of } \Gamma \vdash_{\text{CSYLL}} S = \mathsf{T} \rangle \\
 & H(\Gamma) \vdash_{\text{CSYLL}} H(S) = \mathsf{T}.
\end{array}
$$

Hence, T_{CSYLL} is complete with respect to $K_{\mathcal{L}}$.

Now, we show that T_{CSYLL} gives a decision procedure for CSYLL.

Theorem 5. T_{CSYLL} *is a (complete) decision procedure for CSYLL.*

Proof. We have proved using the CiME system that the equations E_{CSYLL} are confluent and terminating modulo A_{CSYLL}. We use `fa(_)` and `ex(_)` for $[\_]$ and $\{\_\}$, respectively, in the CiME specification of CSYLL.

```
let S_CSYLL =
   signature
   " TRUE,FALSE : constant ;
      not,fa,ex : unary ;
      equ,neq,or,and : AC ;
      imp,cos : infix binary ; ";
let V = vars "A B C";
let R_CSYLL = TRS S_CSYLL V
   "  A equ TRUE -> A ;
      A equ A -> TRUE ;
      not(A) -> A equ FALSE ;
      A neq B -> not (A equ B) ;
      A or A -> A ;
      A or FALSE -> A ;
      A or TRUE -> TRUE ;
      A or (B equ C) -> (A or B) equ (A or C) ;
      A and B -> (A or B) equ A equ B ;
      A imp B -> ( A or B ) equ B ;
      A cos B -> B imp A ;
      fa(A) -> not(ex(not(A))) ;
      ex(TRUE) -> TRUE ;
      ex(FALSE) -> FALSE ;
      ex(A) or ex(B) -> ex( A or B ) ;";
```

For confluence we have:

```
CiME> confluence R_CSYLL ;
Computing self critical pairs
...
System is confluent (160 critical pair(s) tested).
- : bool = true
```

For termination we use the dependency pair method with marks:

```
CiME> termination R_CSYLL ;
Entering the termination expert. Verbose level = 0
The dependency graph is
...
(16 termination constraints)
Search parameters: simple polynomials, coefficient bound is 10.
Solution found for these constraints:
[TRUE] = 0;
[FALSE] = 0;
[not](X0) = X0 + 1;
[fa](X0) = 1;
[ex](X0) = 0;
[equ](X0,X1) = X1 + X0 + 1;
[neq](X0,X1) = X1 + X0 + 2;
[or](X0,X1) = X1*X0 + X1 + X0;
[and](X0,X1) = X1*X0 + 2*X1 + 2*X0 + 2;
[imp](X0,X1) = X1*X0 + 2*X1 + X0 + 1;
[cos](X0,X1) = X1*X0 + X1 + 2*X0 + 1;

Termination proof found.
- : unit = ()
```

Since T_{CSYLL} is complete with respect to $K_{\mathcal{L}}$, it follows that T_{CSYLL} is a complete decision procedure for CSYLL.

Lemma 3. *The* CSYLL-canonical form *of any CSYLL sentence S is either* T, F *or* $S_0 \equiv \cdots \equiv S_n$*, where all* S_i *are distinct and of the form* $\{t_i\}$*, where* t_i *is a DS-canonical form.*

Proof. The proof has again been obtained using Maude's Sufficient Completeness Checker [15] and the intuition behind it is similar to that of DS-canonical forms.

As a consequence, we can use T_{CSYLL} as a decision procedure for CSYLL without losing the good feature of having T and F as canonical forms. That is, we have the following equivalences for any CSYLL expressions S and S':

$$T_{\text{CSYLL}} \vdash S \equiv S' = \mathsf{T} \;\Leftrightarrow\; \mathsf{can}_{E_{\text{CSYLL}}/A_{\text{CSYLL}}}[S \equiv S'] = [\mathsf{T}]$$

and

$$T_{\text{CSYLL}} \vdash S \equiv S' = \mathsf{F} \;\Leftrightarrow\; \mathsf{can}_{E_{\text{CSYLL}}/A_{\text{CSYLL}}}[S \equiv S'] = [\mathsf{F}]\,.$$

By the similarity between CSYLL-canonical forms and DS-canonical forms the problems of extracting models can be addressed similarly.

5.3 Examples

We now present some examples of use of the executable axiomatization of T_{CSYLL} in the Maude System.

Example 1 Recall the 'syllogism' mentioned in the introduction of this paper: 'All men are mortal and all Colombians are men, therefore all Colombians are mortal'. Using the Σ_{CSYLL}-like notation, this is symbolized as follows:

$$[\text{Men} \Rightarrow \text{Mortal}] \wedge [\text{Colombian} \Rightarrow \text{Men}] \;\Rightarrow\; [\text{Colombian} \Rightarrow \text{Mortal}]\,.$$

In Maude, we have:

```
Maude> red [Men imp Mortal] and [Colombian imp Men]
        imp [Colombian imp Mortal] .
reduce in CSYLL : [Men imp Mortal] and [Colombian imp Men]
        imp [Colombian imp Mortal] .
rewrites: 90 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
```

This should be expected since $[Q \Rightarrow R] \wedge [P \Rightarrow Q] \Rightarrow [P \Rightarrow Q]$ is a substitution instance of axiom 3. of $K_{\mathcal{L}}$.

Example 2 This second example presents an execution script in which reductions of axioms 2.-7. are all shown to be equal to T in all the cases.

```
reduce in CSYLL : [A imp A] .
rewrites: 9 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
==========================================
reduce in CSYLL : [B imp C] and [A imp B] imp [A imp C] .
rewrites: 90 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
==========================================
reduce in CSYLL : [B imp C] and {B and A} imp {B and C} .
rewrites: 97 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
==========================================
reduce in CSYLL : {B and A} imp {A and A} .
rewrites: 14 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
==========================================
reduce in CSYLL : not {A and A} imp [A imp B] .
rewrites: 26 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
==========================================
reduce in CSYLL : not [A imp B] equ {A and not B} .
rewrites: 14 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
```

Example 3 Consider the syllogism instance[4]:

'All dogs are mammals and no mammal is a herbivore. Thus, no dog is a herbivore.'

Using Σ_{CSYLL} syntax we have:

$$[\text{Dogs} \Rightarrow \text{Mammals}] \wedge \neg\{\text{Mammals} \wedge \text{Herbivore}\} \;\Rightarrow\; \neg\{\text{Dogs} \wedge \text{Herbivore}\}.$$

Translating this into Maude we get:

```
Maude> red  [Dogs imp Mammals] and
       not {Mammals and Herbivore} imp
       not {Dogs and Herbivore} .
reduce in CSYLL : not {Mammals and Herbivore} and
       [Dogs imp Mammals] imp
        not {Dogs and Herbivore} .
rewrites: 70 in 0ms cpu (0ms real) (~ rewrites/second)
result Sentence: TRUE
```

[4] http://en.wikipedia.org/wiki/Monadic_predicate_logic

Another decision procedure using term rewriting for Syllogistic Logic has been developed by Glashoff [11] using a quite different approach. His approach consists in modeling Aristotle's logic using the Aristotelian rules of deduction as inference rules (i.e. as rewrite rules) while avoiding the use of Boolean connectives. Given an input 'syllogism', this system generates all the possible conclusions from the premises and then checks whether the conclusion of the desired sentece belongs to the generated set: if it does, the syllogism is indeed valid, otherwise it is not. Considering the classification of interests on Syllogistic Logic, Glashoff's approach can be regarded as a pure philosophical approach. In contrast, our approach is merely logical. We have extended a decision procedure for propositional logic and exploited the novel idea of using Boolean connectives and the complement relation to solve the word problem in a more general logic, namely CSYLL. Our approach is analogous to the idea of Łukasiewics of embedding Syllogistic Logic in a more general logic like FOL, except that in our case we show that many-sorted equational logic is sufficient and full FOL is not needed. Furthermore, this work seems a good step towards the more ambitious goal of exploring, following the Dijkstra-Scholten approach, other fragments of FOL which are both decidable and admit an equational representation.

6 Conclusions

We have presented equational axiomatizations of the Dijkstra-Scholten formulations of propositional logic and of a fragment of Monadic First-Order logic, namely Syllogistic Logic with Complements and Boolean connectives. We have explained how these axiomatizations, when specified in Maude, yield decision procedures for these logics. To the best of our knowledge these mechanizations of the Dijkstra-Scholten logic and of CSYLL are new. They can be useful as components of mechanized reasoning systems for program verification in the spirit of Dijkstra.

There are several natural extensions of this work. One, already undertaken in [25], is to compare the equational axiomatization of the Dijkstra-Scholten propositional logic presented here with other decision procedures for propositional logic. Another is to extend the present axiomatizations to larger fragments of the Dijkstra-Scholten logic: first to full MFOL which is still decidable, then to other still decidable segments of FOL, and then to the undecidable case of full First-Order logic not by equations only, but by rewrite theories [22]. A third possible extension is the already-mentioned use of these equational axiomatizations within program verification systems in the Dijkstra style.

References

1. L. Bachmair and N. Dershowitz. Inference rules for rewrite-based first-order theorem proving. In *LICS*, pages 331–337. IEEE Computer Society, 1987.
2. D. Basin, M. Clavel, and J. Meseguer. Rewriting logic as a metalogical framework. *ACM Transactions on Computational Logic*, 5:528–576, 2004.
3. G. S. Boolos, J. P. Burgees, and R. C. Jeffrey. *Computability and Logic.* Cambridge University Press, 4th edition, 2002.
4. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theor. Comput. Sci.*, 236(1-2):35–132, 2000.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
6. M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude - A High-Performance Logical Framework.* Springer LNCS Vol. 4350, 1st edition, 2007.
7. M. Clavel, J. Meseguer, and M. Palomino. Reflection in membership equational logic, many-sorted equational logic, horn logic with equality, and rewriting logic. *Electr. Notes Theor. Comput. Sci.*, 71, 2002.
8. J. Corcoran. Completeness of an ancient logic. *J. Symb. Log.*, 37(4):696–702, 1972.
9. E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics.* Springer-Verlag, 1990.
10. A. Foret. Rewrite rule systems for modal propositional logic. In *Journal of Logic Programming*, volume 12, pages 281–298, 1992.
11. K. Glashoff. Aristotelian syntax from a computational–combinatorial point of view. *Journal of Logic and Computation*, 15:949–937, 2005.
12. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
13. D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math.* Texts and Monographs in Computer Science. Springer Verlag, 1993.
14. D. Gries and F. B. Schneider. Equational propositional logic. *Inf. Process. Lett.*, 53(3):145–152, 1995.
15. J. Hendrix, H. Ohsaki, and J. Meseguer. Sufficient completeness checking with propositional tree automata. Technical Report UIUCDCS-R-2005-2635, University of Illinois Urbana-Champaign, 2005.
16. J. Hsiang. *Topics in automated theorem proving and program generation.* PhD thesis, University of Illinois at Urbana-Champaign, 1982.
17. N. Jacobson. *Basic algebra. I.* W. H. Freeman and Co., San Francisco, Calif., 1974.
18. J.-P. Jouannaud, editor. *Rewriting Techniques and Applications, First International Conference, RTA-85, Dijon, France, May 20-22, 1985, Proceedings*, volume 202 of *Lecture Notes in Computer Science*. Springer, 1985.
19. Laboratoire de Recherche en Informatique. The CiME 2.0 System (`http://cime.lri.fr/`).
20. V. Lifschitz. On calculational proofs. *Ann. Pure Appl. Logic*, 113(1-3):207–224, 2001.
21. J. Łukasiewicz. *Aristotle's Syllogistic, From the Standpoint of Modern Formal Logic.* Oxford University Press, 1951.
22. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

23. L. S. Moss. Syllogistic logic with complements. Draft, 2007.
24. F. Parisi-Presicce, editor. *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, June 1997, Selected Papers*, volume 1376 of *Lecture Notes in Computer Science*. Springer, 1997.
25. C. Rocha and J. Meseguer. Five isomorphic Boolean theories and four equational decision procedures. Technical Report 2007-2818, University of Illinois at Urbana-Champaign, 2007.
26. G. F. Simmons. *Introduction to topology and modern analysis.* McGraw-Hill Book Co., Inc., New York, 1963.