A Tableau Calculus for Quantifier-free Set Theoretic Formulae

Bernhard Beckert[1] and Ulrike Hartmer[2,⋆]

[1] University of Karlsruhe, Institute for Logic, Complexity and Deduction Systems, D-76128 Karlsruhe, Germany. E-mail: `beckert@ira.uka.de`
[2] Deutsche Telekom AG, Technologiezentrum Darmstadt, D-64307 Darmstadt, Germany. E-mail: `hartmer@tzd.telekom.de`

Abstract. Set theory is the common language of mathematics. Therefore, set theory plays an important rôle in many important applications of automated deduction. In this paper, we present an improved tableau calculus for the decidable fragment of set theory called *multi-level syllogistic with singleton* (MLSS). Furthermore, we describe an extension of our calculus for the bigger fragment consisting of MLSS enriched with free (uninterpreted) function symbols (MLSSF).

1 Introduction

Set theory is the common language of mathematics. Therefore, set theory plays an important rôle in many important applications of automated deduction. For example, some of the most widely used specification languages, namely the Z and B specification languages, are completely based on set theory. For other languages, sets are at least a very important construct, frequently used in specifications either on the meta-level or as a data structure of the specified programs. Set theoretic proof obligations occur both as part of proving an implementation to be sound w.r.t. a specification and as part of immanent reasoning (such as consistency checks, proving invariants, pre- and post-conditions).

Set theoretic reasoning, i.e., employing special purpose techniques instead of using the axioms of set theory, is indispensable for automated deduction in real world domains. Automated deduction tools can, for example, be integrated into interactive software verification systems and relieve the user from the need to interactively handle simple set theoretic problems that do not require his or her intuition but merely a combinatorial search.

In this paper, we present an improved tableau calculus for the decidable fragment of set theory called *Multi-level Syllogistic with Singleton* (MLSS). Furthermore, we describe an extension of our calculus for the bigger fragment consisting of MLSS enriched with free (uninterpreted) function symbols (MLSSF).

Multi-level Syllogistic (MLS) consists of quantifier-free formulae built using the set theoretic predicates *membership*, *equality*, *set inclusion*, the binary functions *union*, *intersection*, *set difference*, and a constant representing the empty

⋆ This work was carried out while the author stayed at the University of Karlsruhe.

set. In the extension MLSS of MLS, n-ary functions $\{\cdot\}_n$ can be used to construct singletons, pairs, etc.

The expressiveness of MLSS is sufficient for many applications. MLSS formulae can contain variables, which are implicitly universally quantified. The main restriction is that there is no existential quantification; thus, sentences such as "there is an infinite set" cannot be formalised within MLSS.

Our calculus for MLSS, which is a sound and complete decision procedure, is an extension of the tableau-based calculus for MLSS that Cantone described in [4]. It does not require formulae to be in normal form, whereas Cantone's calculus only contains rules for normalised MLSS literals (which are not allowed to contain complex terms) and relies on a pre-processing transformation for normalising formulae. The handling of free function symbols in the extended calculus for MLSSF employs E-unification techniques for reducing the search space by finding term pairs that, when shown to be equal, close a tableau branch.

Several methods for handling set theory in tableaux or the sequent calculus (without the restriction to a certain fragment) have been proposed: In [2], Brown presents a first-order sequent calculus that contains special rules for many set theoretic symbols. De Nivelle [10] and Pastre [14] introduce sequent calculi for set theory. Shults [15] describes a tableau calculus with special set theoretic rules. All these calculi, however, are incomplete (no semi-decision procedures).

Decision and semi-decision procedures for various extensions of MLS have been described in the literature; these, however, are not based on tableaux but are highly non-deterministic search procedures and are not suitable for implementation; an overview can be found in [5, 6]. Extensions of MLS that are known to be decidable include: MLS with powerset and singleton [3, 7], with relational constructs [9], with unary union [8], and with a choice operator [11].

This paper is structured as follows: In Sect. 2, we define the syntax and semantics of the fragments MLSS and MLSSF of set theory. In Sect. 3, we introduce those parts of our calculus that are not specific for set theoretic formulae. In Sect. 4, we describe our calculus for the fragment MLSS; and in Sect. 5, we extend the calculus for handling the fragment MLSSF including free function symbols. As an example, we present a proof for an MLSSF formula in Sect. 6. Finally, in Sect. 7, we draw conclusions and discuss future work. Due to space restrictions, proofs are not included in this paper; they can be found in [12].

2 Syntax and Semantics

2.1 Syntax of MLSS and MLSSF

We handle two classes of set theoretic formulae: The first are formulae in the fragment *multi-level syllogistic with singletons* (MLSS); this is the quantifier free fragment of set theoretic formulae using (a) the set theoretic predicate symbols $\sqsubset\!\!\!-$ (membership), $\approx$ (equality), $\sqsubseteq$ (set inclusion), (b) the set theoretic function symbols $\sqcap$ (intersection), $\sqcup$ (union), $\setminus$ (set difference), and $\{\cdot\}_n$ with arity $n \geq 1$ (singleton, pair, etc.), and (c) the set theoretic constant $\emptyset$ (the empty set). As usual, the binary function and predicate symbols are written in infix notation, and $\{\cdot\}_n$ is written in circumfix notation.[1] The second fragment, called MLSSF, is the extension of MLSS by free function symbols that have no special set theoretic interpretation.

In the following, we assume that a fixed signature is given consisting of a set *Var* of variables, a set *Const* of constants, and a set *Func* of function symbols.

Definition 1. *The set of* pure set terms *is inductively defined by: (1) All variables $x \in Var$, all constants $c \in Const$, and $\boldsymbol{\emptyset}$ are pure set terms. (2) If t_1, t_2 are pure set terms, then $t_1 \sqcap t_2$, $t_1 \sqcup t_2$, and $t_1 \setminus t_2$ are pure set terms. (3) For all $n \geq 1$, if $t_1, \ldots, t_n$ are pure set terms, then $\{t_1, \ldots, t_n\}_n$ is a pure set term.*

The set of set terms *is inductively defined by: (1) All pure set terms are set terms. (2) If $f \in Func$ is a function symbol of arity $n \geq 1$ and $t_1, \ldots, t_n$ are set terms, then $f(t_1, \ldots, t_n)$ is a set term.*

A set term is called functional *if it is of the form $f(t_1, \ldots, t_n)$.*

Note that functional set terms can contain non-functional set terms (which are not necessarily pure) and vice versa.

MLSS and MLSSF are built using the logical connectives $\vee$ (disjunction), $\wedge$ (conjunction), $\neg$ (negation), and $\rightarrow$ (implication). Formulae that are identical up to associativity of $\vee$ and $\wedge$ are identified.

Definition 2. *If t_1, t_2 are pure set terms (resp. set terms), then $t_1 \mathrel{\underline{\in}} t_2$, $t_1 \approx t_2$, and $t_1 \sqsubseteq t_2$ are* MLSS *(resp.* MLSSF*)* atoms*. If p is an MLSS (MLSSF) atom, then p and $\neg p$ are* MLSS *(*MLSSF*)* literals*.*

The sets of MLSS *and* MLSSF formulae *are inductively defined by: (1) All MLSS (MLSSF) literals are MLSS (MLSSF) formulae. (2) If ϕ, ψ are MLSS (MLSSF) formulae, then $\neg\phi$ and $\phi \rightarrow \psi$ are MLSS (MLSSF) formulae. (3) If $\phi_1, \ldots, \phi_n$ are MLSS (MLSSF) formulae, then $\phi_1 \wedge \cdots \wedge \phi_n$ and $\phi_1 \vee \cdots \vee \phi_n$ are MLSS (MLSSF) formulae ($n \geq 2$).*

To simplify notation, we use the negative versions $\not\mathrel{\underline{\in}}$, $\not\approx$, and $\not\sqsubseteq$ of the predicate symbols $\mathrel{\underline{\in}}$, $\approx$, and $\sqsubseteq$, where $s \not\mathrel{\underline{\in}} t$ is an abbreviation for $\neg(s \mathrel{\underline{\in}} t)$, etc.

2.2 Semantics

We use the semantics of set theory (and thus its fragments MLSS and MLSSF) as it is defined by the ZF axiom system or, equivalently, by the von Neumann hierarchy (cumulative hierarchy) of sets (see for example [13] for a detailed discussion of the semantics of set theory).

Definition 3. *Let Ord denote the class of all ordinals. The* von Neumann hierarchy *of sets is defined by $\mathcal{V} = \bigcup_{\alpha \in Ord} \mathcal{V}_\alpha$ where (1) $\mathcal{V}_0 = \emptyset$, (2) $\mathcal{V}_\alpha = \bigcup_{\beta < \alpha} \mathcal{V}_\beta$ for each limit ordinal α, and (3) $\mathcal{V}_{\alpha+1}$ is the powerset of $\mathcal{V}_\alpha$ for each ordinal α.*

[1] To avoid confusion we use the non-standard symbols $\mathrel{\underline{\in}}, \approx, \sqsubseteq, \sqcap, \sqcup, \boldsymbol{\emptyset}$ on the object level and the standard symbols $\in, =, \subset, \cap, \cup, \emptyset$ on the meta level.

We only define the semantics of the fragment MLSSF; the semantics of MLSS is the same as that of MLSSF for the case of an empty set of function symbols.

Definition 4. *A* set structure $M = \langle D, I \rangle$ *consists of a domain D and an interpretation I with the following properties: The elements of D are sets in the von Neumann hierarchy; D is closed under the set operations $\cap$, $\cup$, $\setminus$, and $\{\cdot\}_n$ ($n \geq 1$), and it contains the empty set; I interprets (1) each constant symbol $c \in Const$ by an element of D, (2) each function symbol $f \in Func$ of arity n by a function $D^n \to D$, (3) the constant $\emptyset$ by the empty set, (4) the predicate symbols by their canonical interpretations, i.e., $\sqsubset$ by $\in$, $\approx$ by the identity relation, and $\sqsubseteq$ by $\subseteq$, (5) the set theoretic function symbols by their canonical interpretations, i.e., $\sqcup$ by $\cup$, $\sqcap$ by $\cap$, $\setminus$ by $\setminus$, and $\{\cdot\}_n$ by $\{\cdot\}_n$ ($n \geq 1$).*

Definition 5. *Given a set structure $M = \langle D, I \rangle$, a* variable assignment *is a mapping $\mu : Var \to D$ from the set of variables to the domain D. The combination of M and a variable assignment μ associates (by structural recursion) with each set term t an element $val_{M,\mu}(t)$ of D; and it associates with each MLSSF formula ϕ either true or false. A formula ϕ is true in M (and M is a model of ϕ) if, for all variable assignments μ, $val_{M,\mu}(\phi) = true$; else ϕ is false in M.*

Definition 6. *An MLSSF formula ϕ is* satisfiable *if there is a set structure M such that ϕ is true in M; ϕ is* valid *if it is true in all set structures.*

3 Tableaux for Quantifier-free Formulae

In this section, we introduce those parts of our calculus that are not specific for set theory. In particular, we define the expansion rules for logical connectives.

The non-literal MLSS and MLSSF formulae are divided into two classes: α for formulae of conjunctive type and β for formulae of disjunctive type. In the left part of Table 1, the expansion rules for α- and β-formulae are given schematically. Premisses and conclusions are separated by a horizontal bar, while vertical bars in the conclusion denote different *extensions*. The tableau expansion rule corresponding to a formula ϕ is obtained by looking up the formula type of ϕ in the right part of Table 1 and instantiating the matching rule schema. The formulae in an extension are implicitly conjunctively connected, and different extensions are implicitly disjunctively connected. We use n-ary α- and β-rules, i.e., when the β-rule is applied to a formula $\psi = \phi_1 \vee \ldots \vee \phi_n$, then ψ is broken up into n subformulae (instead of splitting it into two formulae $\phi_1 \vee \ldots \vee \phi_r$ and $\phi_{r+1} \vee \ldots \vee \phi_n$).

Below, tableaux and tableau proofs are defined in general; which expansion rules (besides those for the logical connectives) and which closure rules are to be used is described in the following sections.

Definition 7. *An* MLSS tableau *(an* MLSSF tableau*) is a finitely branching tree whose nodes are MLSS formulae (MLSSF formulae). A* branch *in a tableau $\mathcal{T}$ is a maximal path in $\mathcal{T}$ (where no confusion can arise, a branch is often identified with the set of formulae it contains).*

Table 1. Rule schemata for α- and β-formulae, and correspondence between non-literal formulae and rule types.

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \vdots\\ \alpha_n\end{array}} \qquad \frac{\beta}{\beta_1 \mid \cdots \mid \beta_n}$$

α	$\alpha_1, \ldots, \alpha_n$
$\phi_1 \wedge \ldots \wedge \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \vee \ldots \vee \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$
$\neg(\phi \rightarrow \psi)$	$\phi,\ \neg\psi$
$\neg\neg\phi$	ϕ

β	$\beta_1, \ldots, \beta_n$
$\phi_1 \vee \ldots \vee \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \wedge \ldots \wedge \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$
$\phi \rightarrow \psi$	$\neg\phi,\ \psi$

Given an MLSS (MLSSF) formula ϕ and a set of tableau expansion rules, the tableaux for ϕ are (recursively) defined by: (1) The tree consisting of a single node labelled with ϕ is a tableau for ϕ (initialisation rule). (2) Let $\mathcal{T}$ be a tableau for ϕ, B a branch of $\mathcal{T}$, and let the premiss of one of the expansion rules occur on B. If the tree $\mathcal{T}'$ is constructed by extending B by as many new linear subtrees as the tableau expansion rule has extensions, where the nodes of the new subtrees are labelled with the formulae in the extensions, then $\mathcal{T}'$ is a tableau for ϕ (expansion rule).

Since the free variables in quantifier-free formulae are implicitly universally quantified, a formula $\phi(x)$ is valid if and only if a *Skolemisation* $\neg\phi(c)$ of its negation is unsatisfiable. Thus, free variables can be eliminated, and a tableau calculus for formulae without free variables is sufficient for checking the validity of a given formula ϕ.

Definition 8. *Given an MLSSF formula $\phi(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are the (free) variables in ϕ ($n \geq 0$), a formula $\phi(c_1, \ldots, c_n)$ is a* Skolemisation *of ϕ if $c_1, \ldots, c_n$ are constants that do not occur in $\phi(x_1, \ldots, x_n)$.*

Definition 9. *A tableau $\mathcal{T}$ is a* tableau proof *for an MLSS/MLSSF formula ϕ, if (1) $\mathcal{T}$ is a tableau for a Skolemisation of $\neg\phi$ (Def. 8), and (2) all branches of $\mathcal{T}$ are closed (Def. 11).*

4 A Tableau Calculus for MLSS

In this section, we present tableau expansion rules that—in combination with the expansion rules for the logical connectives—represent a sound and complete calculus for MLSS, i.e., for formulae built using only *pure* set literals. It can easily be turned into a decision procedure (see Sect. 4.5).

Since the (negation of) the formula to be proven is first Skolemised and is then split into literals using the rules for logical connectives, it is sufficient to define expansion rules for handling pure, variable free set literals.

4.1 Expansion Rules for Splitting Complex Set Terms

The first group of expansion rules applies simple set theoretic lemmata such as "if $s \in t_1 \cup t_2$ then $s \in t_1$ or $s \in t_2$" to (a) eliminate literals containing the set inclusion predicate $\sqsubseteq$ and replace them with (in-)equalities, and to (b) split complex terms on the right side of the membership predicate $\in$ into their constituents. These rules can be described using the α- and β-rule schemata (left part of Table 1); the formula and rule types are listed in Table 2.

Table 2. Rule types for splitting complex set terms.

Name	α	$\alpha_1, \ldots, \alpha_n$
(R1)	$s \sqsubseteq t$	$s \approx s \sqcap t$
(R2)	$s \not\sqsubseteq t$	$s \not\approx s \sqcap t$
(R3)	$s \in t_1 \sqcap t_2$	$s \in t_1,\ s \in t_2$
(R4)	$s \in t_1 \setminus t_2$	$s \in t_1,\ s \notin t_2$
(R5)	$s \notin t_1 \sqcup t_2$	$s \notin t_1,\ s \notin t_2$
(R6)	$s \notin \{t_1, \ldots, t_n\}_n$	$s \not\approx t_1, \ldots, s \not\approx t_n$

Name	β	$\beta_1, \ldots, \beta_n$
(R7)	$s \in t_1 \sqcup t_2$	$s \in t_1,\ s \in t_2$
(R8)	$s \notin t_1 \sqcap t_2$	$s \notin t_1,\ s \notin t_2$
(R9)	$s \notin t_1 \setminus t_2$	$s \notin t_1,\ s \in t_2$
(R10)	$s \in \{t_1, \ldots, t_n\}_n$	$s \approx t_1, \ldots, s \approx t_n$

4.2 Expansion Rules for Handling Equality and Inequality

There are three types of special rules for handling the equality and inequality of sets. First, there are two rules ((EQ1) and (EQ2) in Table 3) that allow to "apply" an equality $t_1 \approx t_2$ to other literals in a very restricted way: an equality can only be applied at the top level and only to the right side of an atom whose predicate symbol is $\in$. That is, an equality can only be applied to derive one of the atoms $s \in t_1$ and $s \in t_2$ from the other one. This restriction is important, because the possibility to apply equalities arbitrarily to other literals would lead to a much larger search space.

Second, it is possible to derive $s_1 \not\approx s_2$ from $s_1 \in t$ and $s_2 \notin t$ (rule (R11) in Table 3). This rule is based on the fact that two objects are different if one of them is an element of some set and the other is not.

Third, the opposite of the above holds as well: if two sets t_1 and t_2 are different, then one of them contains an element c that is not element of the other set. Unfortunately, this leads to a branching rule (rule (R12) in Table 3), because c can be an element of t_1 (and not of t_2) or of t_2 (and not of t_1). A new constant has to be introduced representing the unknown element c.

Table 3. Rules for handling equality and inequality, and the restricted cut rule.

$$\frac{\begin{array}{c} t_1 \approx t_2 \\ s \in t_1 \end{array}}{s \in t_2}$$

(EQ1)

$$\frac{\begin{array}{c} t_1 \approx t_2 \\ s \in t_2 \end{array}}{s \in t_1}$$

(EQ2)

$$\frac{\begin{array}{c} s_1 \in t \\ s_2 \notin t \end{array}}{s_1 \not\approx s_2}$$

(R11)

$$\frac{t_1 \not\approx t_2}{\begin{array}{c|c} c \in t_1 & c \notin t_1 \\ c \notin t_2 & c \in t_2 \end{array}}$$

where c is a constant new to the tableau
(R12)

$$\frac{}{s \in t \mid s \notin t}$$

where s resp. $\{\ldots, s, \ldots\}$ and t resp. $\{\ldots, t, \ldots\}$ are top-level terms on the branch
(Cut)

4.3 The Cut Rule

The cut rule (Table 3) may be applied to extend a tableau branch B using as cut formula atoms $s \sqsubseteq t$ where the set terms s and t occur (a) as top-level arguments of a literal on B, or (b) as arguments on the second level if the top-level function symbol is $\{\cdot\}_n$. In practice the cut rule is rarely needed to find a proof; it is, for example, needed to detect implicit membership cycles on a branch; see Sect. 4.4.

Example 10. If $t_1 \sqsubseteq \{t_2, (t_3 \sqcap t_4)\}$ and $t_5 \sqcap t_6 \approx t_7$ are literals on the branch, then $t_1, t_2, (t_3 \sqcap t_4), (t_5 \sqcap t_6), t_7$ may be used in a cut rule application and t_3, t_4, t_5, t_6 may not be used.

4.4 The Closure Rules

Tableau expansion rules add formulae to a tableau branch being true in all set structures that are models of the expanded branch; the purpose of closure rules is to detect inconsistencies, i.e., formulae on a branch that are false in all set structures. There are four types of inconsistencies that have to be considered: (1) In no set structure both a formula ϕ and its complement $\neg\phi$ are true; thus, a pair $\phi, \neg\phi$ is inconsistent (for completeness it is sufficient to only consider complementary *literals*). (2) No object is an element of the empty set; therefore, a literal of the form $t \sqsubseteq \mathbf{\emptyset}$ is inconsistent. (3) As no object is different from itself, literals of the form $t \not\approx t$ are inconsistent. (4) The existence of a membership cycle, i.e., of sets $u_1, \ldots, u_k$ such that $u_i \in u_{i+1}$ $(1 \leq i < k)$ and $u_k \in u_1$, would contradict the Axiom of Foundation. In fact, there are by construction no sets in the von Neumann hierarchy that form a membership cycle. Thus, literals defining a membership cycle are inconsistent; in particular, $t \sqsubseteq t$ is inconsistent.

Definition 11. *A tableau branch B is* closed *if it contains (1) a complementary pair ϕ and $\neg\phi$ of literals, (2) a literal of the form $t \sqsubseteq \mathbf{\emptyset}$, (3) a literal of the form $t \not\approx t$, or (4) for some $k \geq 1$, literals $t_i \sqsubseteq t_{i+1}$ $(1 \leq i < k)$ and $t_k \sqsubseteq t_1$.*

4.5 Soundness, Completeness, Termination

The calculus for MLSS described in the previous sections is sound and complete:

Theorem 12. *An MLSS formula ϕ is valid iff there is a tableau proof for ϕ using the expansion rules from Tables 1–3 and the closure rule from Def. 11.*

Without further restrictions, the calculus is not a decision procedure. The rule for inequalities ((R12) in Table 3) introduces new constants, and the cut rule can—in connection with rule (R11)—construct new inequalities from the new constants; the interaction of these rules can lead to infinite branches.

Fortunately, the calculus can easily be turned into a decision procedure, observing the fact that chains $c_1, c_2, \ldots$ where c_i is derived applying the inequality rule (R12) to an inequality that contains the constant c_{i-1} cannot be infinite; their length is bounded by the number of (sub-)terms in the initial tableau:

Definition 13. *The* rank *$rank(s)$ of a set term s in a tableau for an MLSS formula ϕ that has been constructed using the expansion rules from Tables 1–3 and the closure rule from Def. 11 is defined as follows: If s occurs in ϕ or has been generated by an application of rules (R1) and (R2), then $rank(s) = 0$; otherwise, i.e., if s is a constant that has been introduced by applying rule (R12) to an inequality $t_1 \not\approx t_2$, then its rank is $rank(s) = 1 + \max\{rank(t_1), rank(t_2)\}$.*

Definition 14. *A tableau $\mathcal{T}$ for an MLSS formula ϕ is* exhausted*, if no tableau expansion rule can be applied to $\mathcal{T}$ without either adding a constant whose rank is greater than the number of (sub-)terms in the root node of $\mathcal{T}$ or adding only formulae to a branch B that already occur on B.*

Theorem 15. *There is an exhausted tableau for an MLSS formula ϕ if and only if ϕ is satisfiable.*

Thus, if a tableau for the Skolemisation of the negation of an MLSS formula ϕ is constructed in a *fair* way (i.e., all possible rule applications are executed sooner or later), then the construction will terminate after a finite number of steps with a tableau that is (a) closed, in which case ϕ is valid, or (b) exhausted, in which case ϕ is not valid.

4.6 Restricting the Search Space

Although it is finite, the search space for a tableau proof is large because of the indeterminism of the cut rule, and because the number of new constants that can be introduced is exponential in the size of the formula to be proven.

Fortunately, it is possible to impose a strong restriction on cut rule applications, which at the same time restricts the number of new constants that are introduced, because a constant c_k of rank k can only be deduced from a constant c_{k-1} of rank $k-1$ after the cut rule has been applied to a literal containing c_{k-1}. The idea is to apply all rules except the cut rule until no further applications are possible, and then to construct a *realisation* of open branches. The realisation of a branch B approximates a model for B (if the branch is satisfiable); it satisfies at least all literals of the form $t_1 \sqsubseteq t_2$ on B. If the realisation does not satisfy all the other literals on B as well, it can be used to find cut rule applications that are (at least potentially) useful.

The switching between the expansion of tableau branches and the construction of possible models, and the way in which we construct models are similar to the method Cantone describes in [4].

Definition 16. *Let $\mathcal{T}$ be a tableau for an MLSS formula ϕ, and let B be a branch of $\mathcal{T}$. Then, G is the set of all (sub-)terms occurring in ϕ; V is the set of all terms $t \in G$ such that $t \sqsubseteq s$ occurs on B and of all constants in ϕ; T is the set of all constants on B that are not in V; $\sim$ is the equivalence relation on $G \cup T$ induced by the equalities on B; T' is the set of all $c \in T$ such that $c \not\sim s$ for all $s \in G$; V' is the set $(V \cup T) \setminus T'$; u_c is, for each $c \in T'$, an element of $\mathcal{V}$ different from all $u_{c'}$ for $c \neq c'$.*

Note, that T' contains the new constants that have been introduced by applying the inequality rule (R12) and that are not equal to other terms (w.r.t. the equalities on the branch). The interpretation of these constants has to be different from the interpretation of all other terms, whereas different terms in V' may have the same interpretation.

Definition 17. *Let B be a branch of a tableau for an MLSS formula ϕ, and let t be a set term on B. Then the* set $P(t)$ of implicit predecessors *of t is defined by: (1) $P(\boldsymbol{\emptyset}) = \emptyset$; (2) $P(c) = \{s \in V \cup T \mid s \sqsubseteq c$ on $B\}$ if $c \in \mathit{Const}$; (3) $P(t_1 \sqcup t_2) = P(t_1) \cup P(t_2)$; (4) $P(t_1 \sqcap t_2) = P(t_1) \cap P(t_2)$; (5) $P(t_1 \setminus t_2) = P(t_1) \setminus P(t_2)$; and (6) $P(\{t_1, \ldots, t_n\}_n) = \{s \in V \cup T \mid s \sqsubseteq \{t_1, \ldots, t_n\}_n$ on $B\} \cup \{t_1, \ldots, t_n\}$.*

The sets of implicit predecessors can be used to detect implicit membership cycles. If, for example, $s \in P(t), t \in P(s)$ for some terms s, t, then the branch can be closed, and it is not necessary to apply expansion rules (especially the cut rule) to make the cycle explicit. Thus, using the predecessor sets we can add another closure rule:

Definition 18. *A tableau branch B is* closed *if it is (a) closed according to Def. 11 or (b) its sets of implicit predecessors contain a cycle, i.e., there are (sub-)terms $t_1, \ldots, t_n$ on B such that $t_1 \in P(t_2)$, ..., $t_{n-1} \in P(t_n)$, $t_n \in P(t_1)$.*

The set $P(t)$ of implicit predecessors contains those terms denoting elements of t whose membership can be deduced from literals on B of the form $s \sqsubseteq a$ (where $a \in \mathit{Const}$) and applying the definition of the set operators. The *realisation* of a branch goes beyond that: it is a partial definition of a set interpretation (different terms may be interpreted by the same set).

Definition 19. *Let B be a branch of a tableau for an MLSS formula ϕ, and let t be a set term on B. If B is not closed (Def. 18), then the* realisation $\mathcal{R}$ *of B is defined by:*[2] *(1) $\mathcal{R}(t) = \emptyset$ if $t = \boldsymbol{\emptyset}$, (2) $\mathcal{R}(t) = \{\mathcal{R}(s) \mid s \in P(t)\} \cup \{u_t\}$ if $t \in T'$, and (3) $\mathcal{R}(t) = \{\mathcal{R}(s) \mid s \in P(t)\}$ otherwise.*

The realisation can be effectively computed and can be used to restrict the application of the cut rule: provided B is exhausted w.r.t. all other expansion rules, the cut rule has only to be applied to terms occurring in literals which are not satisfied by the realisation of B (if there is no such literal, then B is satisfiable and we are done). If, for example, $t_1 \not\sqsubseteq t_2$ occurs on B but $\mathcal{R}(t_1) \in \mathcal{R}(t_2)$, then there has to be a term s such that (a) $\mathcal{R}(s) = \mathcal{R}(t_1)$, i.e., the realisation of s is the same as that of t_1, and (b) s is an implicit member of t_2, i.e., $s \in P(t_2)$—but that membership is not (yet) made explicit on the branch (there is no literal $s \sqsubseteq t_2$ on B). In that case, the cut rule is applied to the literal $s \sqsubseteq t_2$.

Now everything is at hand to define the restricted version of the cut rule:

[2] One has to make sure that the u_c's are different from $\mathcal{R}(t)$ for all terms t; it is always possible to choose such u_c's.

Definition 20. *The* restricted cut rule *(Cut') is identical to rule (Cut) in Table 3 with the exception that (1) it may only be applied to extend a tableau branch B that is not closed (Def. 18) and is exhausted w.r.t. all other expansion rules; and (2) it may only be applied to a cut formula $s \sqsubseteq t$ satisfying one of the following conditions*

- *$t \approx t'$ is on B, $\mathcal{R}(t) \neq \mathcal{R}(t')$, and (a) $s \in P(t)$, $s \notin P(t')$, $s \not\sqsubseteq t$ is* not *on B, or (b) $s \in P(t')$, $s \notin P(t)$, and $s \sqsubseteq t'$ is* not *on B;*
- *$t \not\approx t'$, $c \not\sqsubseteq t$, and $c \sqsubseteq t'$ are on B (for some constant c), $\mathcal{R}(t) = \mathcal{R}(t')$, $\mathcal{R}(s) = \mathcal{R}(c)$, $s \in P(t)$, $s \notin P(t')$, and $s \sqsubseteq t$ is* not *on B;*
- *$t' \not\sqsubseteq t$ is on B, $\mathcal{R}(t') \in \mathcal{R}(t)$, $\mathcal{R}(s) = \mathcal{R}(t')$, $s \in P(t)$, and $s \sqsubseteq t$ is* not *on B.*

Using the restricted version of the cut rule preserves completeness:

Theorem 21. *An MLSS formula ϕ is valid if and only if there is a tableau proof for ϕ using the expansion rules from Tables 1–3 with the restriction of the cut rule according to Def. 20, and the closure rule from Def. 18.*

4.7 A Comparison with Cantone's Calculus

The calculus for MLSS described in the previous sections is similar to that presented by Cantone in [4]. The main difference is that Cantone's calculus is restricted to *normalised* literals, i.e., literals not containing complex set terms:

Definition 22. *A set literal ϕ is* normalised *iff it is of the form $a \sqsubseteq b$, $a \not\sqsubseteq b$, $a \approx b$, $a \not\approx b$, $a \approx b \sqcup c$, $a \approx b \sqcap c$, $a \approx b \setminus c$, or $a \approx \{b_1, \ldots, b_n\}_n$ ($n \geq 1$), where a, b, c and $b_1, \ldots, b_n$ are constants.*

There is a satisfiability preserving transformation of any finite set Γ of set literals into a set of normalised set literals by introducing new constants for complex set terms. For example, $a \sqsubseteq (b \sqcap b')$ is replaced by $c \approx (b \sqcap b')$ and $a \sqsubseteq c$ where c is a new constant. The overhead for computing the transformation is negligible, because its complexity is polynomial in the size of the set to be transformed. However, the introduction of new constants leads to a much bigger search space, even more so as all these new constants occur in equalities.

Our rules (R7), (R3), (R4), and (R10) are—in combination with rules (EQ1) and (EQ2) extensions for handling literals with *complex* set terms of the corresponding rules in Cantone's calculus. For example, our rule (R3), that allows to derive $a \sqsubseteq b$ and $a \sqsubseteq b'$ from $a \sqsubseteq (b \sqcap b')$, corresponds to Cantone's rule that allows to derive $a \sqsubseteq b$ and $a \sqsubseteq b'$ from $c \approx (b \sqcap b')$ and $a \sqsubseteq c$ (for all a, b, c).

There are no rules in Cantone's calculus corresponding to our rules (R5), (R8), and (R9) for literals expressing negated membership. Consider the three literals $\phi = c \not\sqsubseteq (b_1 \sqcup b_2) \setminus b_3$, $\psi_1 = c \sqsubseteq b_1$, and $\psi_2 = c \not\sqsubseteq b_3$, whose conjunction is inconsistent. To close a branch containing these literals, our rules (R9) and (R5) are applied to split the literal ϕ and derive that one of $\neg\psi_1$ and $\neg\psi_2$ holds, thus closing the two resulting sub-branches. Since no rules for splitting ϕ exist in Cantone's calculus, instead rules for positive membership literals have to be used to derive $\neg\phi$ from ψ_1 and ψ_2: first, ϕ has to be normalised, the result are the literals $c \not\mathrel{\sqsubset\!\!\!\!-} d_1$, $d_1 \approx d_2 \setminus b_3$, and $d_2 \approx b_1 \sqcup b_2$ where d_1 abbreviates $(b_1 \sqcup b_2) \setminus b_3$ and d_2 abbreviates $b_1 \sqcup b_2$. Then, with two rule applications, $c \mathrel{\sqsubset\!\!\!\!-} d_2$ and $c \mathrel{\sqsubset\!\!\!\!-} d_1$ are derived. The latter literal can be used to close the branch; it corresponds to the non-normalised literal $\neg\phi$.

The need (and possibility) to derive more complex terms from simpler ones leads to a larger search space. Our rules, that split complex terms into simpler ones, are more goal directed.

5 A Tableau Calculus for MLSSF

5.1 A Simple Extension of MLSS

To extend the calculus described in the previous sections from MLSS to MLSSF, it suffices to (a) relax the restrictions on the equality rules ((EQ1') and (EQ2') in Table 4), and (b) add a cut rule that uses equalities as cut formulae ((Cut') in Table 4). The new rules only need to be applied to functional set terms. Non-functional terms, even if they are not pure, can be handled by the MLSS rules. The result of using these additional rules is a sound and complete calculus for MLSSF; it is, however, not a decision procedure.

Table 4. Additional expansion rules for MLSSF.

$$\frac{\begin{array}{c} s \approx t \\ \phi[s] \end{array}}{\phi[t]} \qquad \frac{\begin{array}{c} t \approx s \\ \phi[s] \end{array}}{\phi[t]}$$

where the occurrence of s in ϕ is inside a functional term
(EQ1') (EQ2')

$$\frac{}{t_1 \approx t_2 \mid t_1 \not\approx t_2}$$

where t_1, t_2 occur on the branch and at least one is a functional term
(Cut')

Theorem 23. *An MLSSF formula ϕ is valid iff there is a tableau proof for ϕ using the expansion rules from Tables 1–4, and the closure rule from Def. 11.*

5.2 Using Rigid *E*-Unification to Restrict the Equality Cut Rule

The additional rules for MLSSF introduced in the previous section are highly non-deterministic. In this section, we describe an expansion rule for MLSSF that is much more goal-directed and leads to a smaller search space. It is based on the concept of *rigid E-unification.*

Definition 24. *A* rigid E-unification problem $\langle E, s, t\rangle$ *consists of a finite set E of equalities and terms s and t; the equalities in E and the terms s and t may contain free variables (and may have variables in common). A substitution σ is a* solution *to the problem iff $E\sigma \models (s\sigma \approx t\sigma)$ where the free variables in $E\sigma$ are "held rigid", i.e., treated as constants.*

The problem of deciding whether a given rigid E-unification problem has a solution is decidable (it is NP-complete). In general, the number of solutions is infinite. An overview of methods for rigid E-unification can be found in [1].

The basic idea is to use rigid E-unification for handling the functional part of formulae on a branch and to use the tableau rules for handling the non-functional (i.e. set theoretic) part. The additional tableau rule we describe in the following forms the connecting link between the two parts.

Consider, for example, a branch B containing the two literals $f(a) \approx b$ and $g(f(a \sqcap (b \sqcup a))) \sqsubseteq g(b)$. They are inconsistent, because $a \cap (b \cup a) = a$ and, thus, $g(f(a \cap (b \cup a))) = g(f(a)) = g(b)$; this implies $g(b) \in g(b)$, which is a membership cycle. To close the branch, one first has to find out what the important set theoretic identities are that have to be proven[3], in this case $a \cap (b \cup a) = a$. It is impossible to do this using only heuristics; here, for example, it is futile to try to show that $a \cap (b \cup a) = b$.

The question of which set theoretic identities have to be proven to close the branch is transformed into rigid E-unification problems as follows: for each pair s, t of terms that, if they were identical would allow to close the branch (e.g. if $s \not\approx t$ is on B), one rigid E-unification problem is generated. In s and t all maximal non-functional sub-terms are replaced by (new) variables; the resulting terms t^x and s^x and the equalities on the branch form a rigid E-unification problem. Each solution to the problem corresponds to identities between non-functional sub-terms that, when proven, allow to close the branch. The corresponding inequalities are (disjunctively connected) added to the branch.

Definition 25. *Given a set L of set literals, the set L^x is constructed by replacing all non-functional (sub-)terms t in L by a new variable x_t. Let the substitution τ_L be defined by: $\tau(x_t) = t$ for all terms t in L that have been replaced (i.e., τ_L is the inverse of the transformation that turns L into L^x: $\tau(L^x) = L$).*

Example 26. If $L = \{(a \sqcap c) \sqcup b \approx c,\ f(c) \sqsubseteq g(a \sqcap c, f(d \setminus e))\}$, then the result of the transformation is $L^x = \{x_1 \approx x_2, f(x_2) \sqsubseteq g(x_3, f(x_4))\}$.

Definition 27. *The* rigid E-unification expansion rule *(EU) is defined as follows: Let B be a branch in a tableau for an MLSSF formula, and let L_B be the set of all literals on B of the form $t_1 \approx t_2$, $t_1 \sqsubseteq t_2$, or $t_1 \not\sqsubseteq t_2$. Let E^x_B be the set of all equalities in L^x_B. Further let $\mu = \{x_1 \leftarrow r_1, \dots, x_n \leftarrow r_n\}$ ($n \geq 1$) be a solution to (1) a rigid E-unification problem $\langle E^x_B, \langle s_1, t_1\rangle, \langle s_2, t_2\rangle\rangle$ such that $s_1 \sqsubseteq t_1$ and $s_2 \not\sqsubseteq t_2$ are in L^x_B or (2) a rigid E-unification problem $\langle E^x_B, \langle t_1, \dots, t_n\rangle, \langle t'_1, \dots, t'_n\rangle\rangle$ such that literals $t_1 \sqsubseteq t'_2, \dots, t_{n-1} \sqsubseteq t'_n$, and $t_n \sqsubseteq t'_1$ in L^x_B form a potential membership cycle. Then B may be extended by n new linear subtrees where the nodes of the new subtrees are labelled with the literals $\tau_{L_B}(x_1) \not\approx \tau_{L_B}(r_1)$, $\dots$, $\tau_{L_B}(x_n) \not\approx \tau_{L_B}(r_n)$.*

Example 28. We continue the example from the beginning of this section and apply the rule (EU) to show that a branch containing the literals $f(a) \approx b$ and

[3] An identity is proven by using it as a cut formula; after the branch that contains its negation has been closed, it is available on the remaining open branch.

$g(f(a \sqcap (b \sqcup a))) \sqsubseteq g(b)$ is inconsistent. The only rigid E-unification problem that can be extracted from these literals is $\langle \{f(x_a) \approx x_b\}, g(f(x_{a\sqcap(b\sqcup a)})), g(x_b)\rangle$. Its simplest solution is the substitution $\{x_a \leftarrow x_{a\sqcap(b\sqcup a)}\}$. Thus, the rule (EU) allows to add $a \not\approx a \sqcap (b \sqcup a)$ to the branch. The complete proof is shown in Fig. 1.

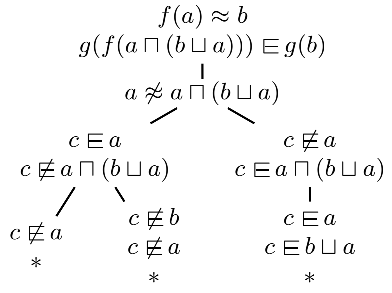
$f(a) \approx b$
$g(f(a \sqcap (b \sqcup a))) \sqsubseteq g(b)$
$a \not\approx a \sqcap (b \sqcup a)$
$c \sqsubseteq a$
$c \not\sqsubseteq a \sqcap (b \sqcup a)$
$c \not\sqsubseteq a$
$c \sqsubseteq a \sqcap (b \sqcup a)$
$c \not\sqsubseteq a$
*
$c \not\sqsubseteq b$
$c \not\sqsubseteq a$
*
$c \sqsubseteq a$
$c \sqsubseteq b \sqcup a$
*

Fig. 1. A tableau proof using the rule (EU) (Example 28).

It is not necessary to consider rigid E-unification problems constructed from inequalities $s \not\approx t$ because, when rule (R11) has been applied, L_B^x contains literals $x_c \sqsubseteq s, x_c \not\sqsubseteq t$ or $x_c \not\sqsubseteq s, x_c \sqsubseteq t$.

The (EU) expansion rule partly overlaps with other expansion rules. It allows, for example, to derive $s_1 \not\approx s_2$ from $s_1 \sqsubseteq t$ and $s_2 \not\sqsubseteq t$ if s_1 and s_2 are non-functional set terms. This is also possible applying the rule (R11).

Theorem 29. *An MLSSF formula ϕ is valid if and only if there is a tableau proof for ϕ using the expansion rules from Tables 1–4, the rule (EU) (Def. 27), and the closure rule from Def. 11.*

The rule (EU) is sound and helps to reduce the search space; we conjecture that completeness is preserved if the rules (EQ1'), (EQ2'), and (Cut') are replaced by (EU), but have not proven this yet.

6 An Example

As an example, we proof that the MLSSF formula

$$\phi \;=\; \big[x \sqsubseteq [(f(x) \setminus f(x \sqcup (y \sqcap x))) \sqcup z \sqcup w] \;\wedge\; w \sqcup y \sqsubseteq x\big] \;\rightarrow\; x \sqsubseteq z$$

is valid; it contains the free function symbol f. Intuitively, the reason for the validity of ϕ is the following: We assume that x is an element of (at least) one of the three sets $u = f(x) \setminus f(x \cup (y \cap x))$, z, and w, and that $w \cup y$ is an element of x. Now, the set u cannot contain x, because $x = (x \cup (y \cap x))$ and therefore u is empty for all interpretations of f; the set w cannot contain x, otherwise there would be a membership cycle $x \in (w \cup y) \in x$. Therefore, z contains x.

Figure 2 shows a tableau proof for ϕ. Its root is labelled with the Skolemisation $\neg \left[a \sqsubseteq [(f(a) \setminus f(a \sqcup (b \sqcap a))) \sqcup d \sqcup e] \;\wedge\; e \sqcup b \sqsubseteq a\right] \;\rightarrow\; a \sqsubseteq d$ of $\neg\phi$. The i-th formula in the tableau is labelled with $[i; j; R]$, which indicates that it has been derived from the j-th formula applying the expansion rule R.

Formula 9 is derived from formulae 7 and 8 applying the E-unification rule. A solution to the E-unification problem $\langle \emptyset, \langle x_a, x_a \rangle, \langle f(x_a), f(x_{a \sqcup (b \sqcap a)}) \rangle \rangle$, which is constructed from 7 and 8, is the substitution $\{x_a \leftarrow x_{a \sqcup (b \sqcap a)}\}$. Accordingly, the inequality $a \not\approx a \sqcup (b \sqcap a)$ is added to the branch.

The branch ending in formula 21 is closed by the membership cycle $e \sqcup b \sqsubseteq a$ and $a \sqsubseteq e \sqcup b$ (formulae 2 and 21). All other branches are closed by complementary literals; their leaves are labelled with the numbers of the closing literals.

If the closure rule that uses the sets of implicit predecessors to detect implicit membership cycles is used (Def. 18), the cut rule application that generates formulae 22 and 23 is not needed. Instead, the branch ending in the literal 21 is already closed; it contains an implicit cycle because $a \sqsubseteq e$ implies $a \sqsubseteq e \sqcup b$ (this cycle is made explicit by the cut rule application).

Implicit cycles can be detected by calculating the predecessor relation for the branch. The set of possible predecessors for the branch ending in formula 21 is $\{a, b, d, e, f(a), f(a \sqcup (b \sqcap a)), (e \sqcup b)\}$. The predecessor sets of the constants are $P(a) = \{e \sqcup b\}$, $P(b) = \emptyset$, $P(d) = \emptyset$, and $P(e) = \{a\}$. The predecessor set of $e \sqcup b$ is $P(e \sqcup b) = P(e) \cup P(b) = \{a\}$. Thus, we have $a \in P(e \sqcup b)$ and $e \sqcup b \in P(a)$, which indicates the presence of an implicit membership cycle.

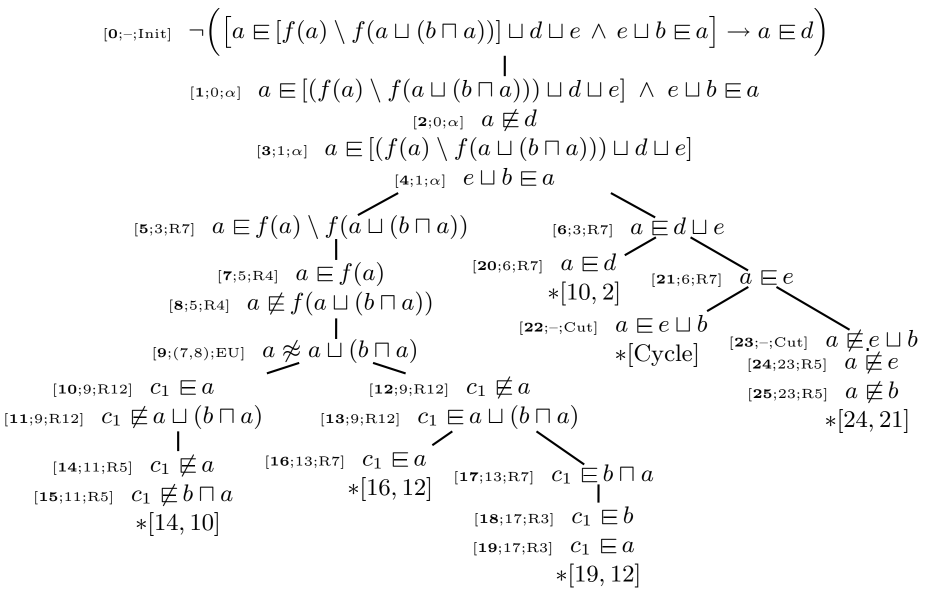
[0;–;Init] $\neg\Big(\big[a \sqsubseteq [f(a) \setminus f(a \sqcup (b \sqcap a))] \sqcup d \sqcup e \;\wedge\; e \sqcup b \sqsubseteq a\big] \rightarrow a \sqsubseteq d\Big)$
[1;0;α] $a \sqsubseteq [(f(a) \setminus f(a \sqcup (b \sqcap a))) \sqcup d \sqcup e] \;\wedge\; e \sqcup b \sqsubseteq a$
[2;0;α] $a \not\sqsubseteq d$
[3;1;α] $a \sqsubseteq [(f(a) \setminus f(a \sqcup (b \sqcap a))) \sqcup d \sqcup e]$
[4;1;α] $e \sqcup b \sqsubseteq a$
[5;3;R7] $a \sqsubseteq f(a) \setminus f(a \sqcup (b \sqcap a))$
[6;3;R7] $a \sqsubseteq d \sqcup e$
[7;5;R4] $a \sqsubseteq f(a)$
[8;5;R4] $a \not\sqsubseteq f(a \sqcup (b \sqcap a))$
[9;(7,8);EU] $a \not\approx a \sqcup (b \sqcap a)$
[10;9;R12] $c_1 \sqsubseteq a$
[11;9;R12] $c_1 \not\sqsubseteq a \sqcup (b \sqcap a)$
[14;11;R5] $c_1 \not\sqsubseteq a$
[15;11;R5] $c_1 \not\sqsubseteq b \sqcap a$
$*[14, 10]$
[12;9;R12] $c_1 \not\sqsubseteq a$
[13;9;R12] $c_1 \sqsubseteq a \sqcup (b \sqcap a)$
[16;13;R7] $c_1 \sqsubseteq a$
$*[16, 12]$
[17;13;R7] $c_1 \sqsubseteq b \sqcap a$
[18;17;R3] $c_1 \sqsubseteq b$
[19;17;R3] $c_1 \sqsubseteq a$
$*[19, 12]$
[20;6;R7] $a \sqsubseteq d$
$*[10, 2]$
[21;6;R7] $a \sqsubseteq e$
[22;–;Cut] $a \sqsubseteq e \sqcup b$
$*[\text{Cycle}]$
[23;–;Cut] $a \not\sqsubseteq e \sqcup b$
[24;23;R5] $a \not\sqsubseteq e$
[25;23;R5] $a \not\sqsubseteq b$
$*[24, 21]$

Fig. 2. Tableau proof for the formula ϕ from Sect. 6.

7 Conclusion

We have presented an improved tableau calculus for the fragment MLSS of set theory that extends the calculus described in [4]. Our tableau expansion rules are more goal-directed; this leads to a smaller search space, which is important for the efficiency of an implementation. Our calculus is a sound and complete decision procedure for MLSS. In addition, we have described a version of the calculus for the larger fragment MLSSF (MLSS with free function symbols); and we have shown how to use a special tableau rule based on *rigid E-unification* to reduce the search space in the case of MLSSF.

Future work includes, besides an implementation and practical evaluation of our calculus, its extension to larger (and undecidable) fragments that (a) contain additional set theoretic operators such as, for example, the power set operator, and that (b) allow existential quantification of variables.

Acknowledgement. We thank Domenico Cantone, Sebastiano Battiato, and three anonymous referees for useful comments on an earlier version of this paper.

References

1. B. Beckert. Rigid E-unification. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume I. Kluwer, 1998. To appear.
2. F. M. Brown. Towards the automation of set theory. *J. of AI*, 10:218–316, 1978.
3. D. Cantone. Decision procedures for elementary sublanguages of set theory. X. *Journal of Automated Reasoning*, 7:193–230, 1991.
4. D. Cantone. A fast saturation strategy for set-theoretic tableaux. In *Proceedings, TABLEAUX, Pont-a-Mousson, France*, LNCS 1227, pages 122–137. Springer, 1997.
5. D. Cantone and A. Ferro. Techniques of computable set theory with applications to proof verification. *Comm. on Pure and Applied Mathematics*, 48:901–946, 1995.
6. D. Cantone, A. Ferro, and E. Omodeo. *Computable Set Theory.* Oxford University Press, 1989.
7. D. Cantone, A. Ferro, and T. J. Schwartz. Decision procedures for elementary sublanguages of set theory. VI. *Comm. on Pure and Applied Mathematics*, 38:549–571, 1985.
8. D. Cantone, A. Ferro, and T. J. Schwartz. Decision procedures for elementary sublanguages of set theory. V. *J. of Computer and Syst. Sciences*, 34:1–18, 1987.
9. D. Cantone and T. J. Schwartz. Decision procedures for elementary sublanguages of set theory. XI. *Journal of Automated Reasoning*, 7:231–256, 1991.
10. H. de Nivelle. Implementation of sequent calculus and set theory. Draft, Feb. 1997.
11. A. Ferro and E. Omodeo. Decision procedures for elementary sublanguages of set theory. VII. *Communications on Pure and Applied Mathematics*, 40:265–280, 1987.
12. U. Hartmer. Erweiterung des Tableaukalküls mit freien Variablen um die Behandlung von Mengentheorie. Diplomarbeit, Universität Karlsruhe, 1997.
13. T. Jech. *Set Theory.* Academic Press, New York, 1978.
14. D. Pastre. Automatic theorem proving in set theory. *J. of AI*, 10:1–27, 1978.
15. B. Shults. Comprehension and description in tableaux. Draft, May 1997.