© Springer, pp. 690-695, LNCS/LNAI 2083, IJCAR 2001

Hilberticus - a Tool Deciding an Elementary Sublanguage of Set Theory

Jörg Lücke

Univ. Dortmund, Fachbereich Informatik, LS 5, 44227 Dortmund, Germany
`luecke@ls5.cs.uni-dortmund.de`, `http://www.hilberticus.de`

Abstract. We present a tool deciding a fragment of set theory. It is designed to be easily accessible via the internet and intuitively usable by anyone who is working with sets to describe and solve problems. The tool supplies features which are well-suited for teaching purposes as well. It offers a self explaining user interface, a parser reflecting the common operator bindings, parse tree visualization, and the possibility to generate Venn diagrams as examples or counterexamples for a given formula. The implemented decision procedure which is based on the semantics of class theory is particularly suitable for this.

Keywords: Set Theory, Decision Procedures, First-Order Logic.

1 Motivation

The language of set theory is one of the most common formal languages. It is used in research fields ranging from mathematics, computer science over natural science and engineering up to economy. This success is probably due to the fact that set theory can be used informally in most of its applications and that the well-known Venn diagrams can illustrate relations and combinations of sets. In contrast to this, in logics the precise definition of the syntax and semantics of languages of set theory is crucial and it provides methods which allow the automatic decision whether formulas of set theory are true, satisfiable, or inconsistent. However, the number of actual users of such methods is far smaller than the number of users of set theory. This is probably the case because an average user of the language of set theory is often not aware of such methods and because existing implementations are not easy to find, to install, and often difficult to learn even if the user should be familiar with the notation of logic. The Hilberticus tool is designed to overcome these problems and it supplies the user with a graphical feedback in the form of Venn diagrams which eases its use and illustrates examples and counterexamples of a satisfiable formula in a convenient way. An alpha version of the tool is accessible via `www.hilberticus.de`. The name *Hilberticus* is a synthesis of the names *Hilbert* and *abacus.*

2 Syntax and semantics of the language SL

Syntax of **SL**. The language **SL** (**S**et **L**anguage) contains the parentheses symbols ')' and '(', the logical connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$, the predicate symbols $=$, $\subseteq$, and $\subset$, the function symbols $\cap$, $\cup$, and $\setminus$, the constant symbols $\emptyset$ and $Đ$, and a countable set of class variables A, B etc. The formulas of **SL** are defined in the usual way. A string $\mathtt{t}$ of the above symbols is called a *term* if $\mathtt{t}$ is a constant symbol, if $\mathtt{t}$ is a variable, or if $\mathtt{t}$ is of the form $(\mathtt{t}_1 \bullet \mathtt{t}_2)$ where $\mathtt{t}_1$ and $\mathtt{t}_2$ are terms and where $\bullet$ denotes a function symbol. A string Φ is called an *atomic formula* if Φ is of the form $(\mathtt{t}_1 = \mathtt{t}_2)$, $(\mathtt{t}_1 \subseteq \mathtt{t}_2)$, or $(\mathtt{t}_1 \subset \mathtt{t}_2)$ (where $\mathtt{t}_1$ and $\mathtt{t}_2$ are terms). Finally, we call Φ a *formula* if it is a propositional combination of atomic formulas. To write and read **SL**-formulas more conveniently we allow the suppression of parentheses using binding priorities for function symbols and logical connectives. The binding of the function symbols is defined according to the list $\cap$, $\cup$, $\setminus$ which is ordered descendingly by binding priority. Similarly, the binding of the logical connectives is defined according to the list $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, where $\Rightarrow$ and $\Leftrightarrow$ have the same priority and associate to the right, in contrast to all other binary symbols.

Semantics of **SL***.* The semantics of **SL** is taken from [1, 2]. There a constructive definition of set theory is developed which is based on the Zermelo-Fraenkel axioms. We denote by $\sigma : V \rightarrow \mathcal{D}$ an *assignment* from the set of variables V into the collection of all classes $\mathcal{D}$. The interpretation I of the predicate and function symbols $=$, $\subseteq$, $\subset$ and $\cap$, $\cup$, $\setminus$ is standard. The constant symbols $\emptyset$ and $\not{D}$ are interpreted as empty and universal class respectively. The domain $\mathcal{D}$ of all classes and the interpretation I make up the *model* of interest which we will denote by $\mathcal{M}$. A formula Φ is *satisfiable* in $\mathcal{M}$ if it is true for some assignment σ and Φ is *true* in $\mathcal{M}$ if it is true for all assignments σ.

3 The decision procedure and its implementation

The decision procedure consists of two steps. Firstly, a formula of **SL** is transformed to a formula of the more fundamental language **SBL** (**S**et **B**asis **L**anguage) using the calculus **L∈** of [1, 2]. Secondly, the formula in **SBL** is decided via the transformation to a propositional formula. The language **SBL** is a first-order language without function symbols and with '$\in$' as the only (uninterpreted) predicate symbol. The calculus **SBL** is the usual predicate calculus for this signature and we call a formula *valid* if it is true in all models of **SBL**.

The calculus **L∈***.* Descriptions of set theory are typically based on a first-order language with '$\in$' and '$=$' as the only predicate symbols. As soon as the formulas of this first-order language are becoming large and unreadable abbreviations are introduced and this not only for formulas but also for terms denoting sets or set-like objects. The *abstraction terms* '$\{v \mid \Phi\}$' are used for this purpose. Generally, the use of abstraction terms is only informal which can lead to problems (as discussed in [1]). In **L∈** such problems are avoided by the introduction of abstraction terms as a part of the language together with axioms and inference rules for their manipulation. A number of theorems show the equivalence of set theories based on **L∈** and set theories based on a usual predicate logic. In the following we focus on some elements of **L∈** and formulate relevant results in a way appropriate to the scope of this paper. For a thorough investigation of **L∈** (including the notation of frames) we refer to [1, 2]. For our purposes the following derivations are of interest [2]:

$$\underset{\mathbf{L}\in}{\vdash}\ \{v|\Phi\} \in \mathtt{t} \Leftrightarrow \exists w\,(w{=}\{v|\Phi\} \wedge w \in \mathtt{t}),$$
$$\underset{\mathbf{L}\in}{\vdash}\ \mathtt{t} \in \{v|\Phi(v)\} \Leftrightarrow \Phi(\mathtt{t}), \quad \underset{\mathbf{L}\in}{\vdash}\ \mathtt{t}_1{=}\mathtt{t}_2 \Leftrightarrow \forall v\,(v \in \mathtt{t}_1 {\Leftrightarrow} v \in \mathtt{t}_2) \tag{1}$$

By successively applying these formulas it is always possible to eliminate all abstraction terms of an **L∈**-formula in favor of an equivalent **SBL**-formula. The fact that the additional axioms and rules for the abstraction terms do only serve as abbreviations in our case is reflected by the following theorem.

Theorem 1. *If Φ is a formula in* **L∈** *and $\tilde{\Phi}$ the formula in* **SBL** *derived by the application of the formulas of (1) to Φ, then Φ is derivable in* **L∈** *iff $\tilde{\Phi}$ is derivable in* **SBL**,

$$\underset{\mathbf{L}\in}{\vdash}\ \Phi \quad \textit{iff} \quad \underset{\mathbf{SBL}}{\vdash}\ \tilde{\Phi}. \tag{2}$$

We can now use the calculus **L∈** to introduce the following abbreviations:

$$\begin{array}{ll} A{\subseteq}B \Leftrightarrow_{def} \forall x\,(x{\in}A \Rightarrow x{\in}B), & A{\cup}B =_{def} \{y|\ y{\in}A \vee y{\in}B\}, \\ A{\subset}B \Leftrightarrow_{def} A{\subseteq}B{\wedge}\neg(A{=}B), & A{\cap}B =_{def} \{y|\ y{\in}A \wedge y{\in}B\}, \\ \emptyset =_{def} \{x|\neg(x = x)\},\ \mathit{Đ} =_{def} \{x|\ x = x\}, & A\backslash B =_{def} \{y|\ y{\in}A \wedge \neg(y{\in}B)\} \end{array} \tag{3}$$

To translate a formula in **SL** to a formula in **SBL** we replace the predicates and functions of **SL** by the corresponding definitions and subsequently apply the formulas of (1) as shown below. The semantics of **SL** (Sect.2) is the same as the semantics of **L∈** defined in [1]. Due to the soundness and completeness of the calculus **L∈** [1] a formula Φ of **SL** is thus true in the underlying model $\mathcal{M}$ if it is derivable in **L∈**. Together with (1) and the soundness and completeness of the predicate calculus we obtain the result:

Theorem 2. *If Φ is a formula in* **SL** *and $\tilde{\Phi}$ the formula derived after the expansion of Φ by the use of the abbreviations of (3) and the successive elimination of abstraction terms by the use of the formulas of (1) then Φ is true in $\mathcal{M}$ iff $\tilde{\Phi}$ is valid in* **SBL**.

Theorem 2 reduces the decidability problem of **SL** to the decidability problem of the calculus **SBL**.

Transformation to a propositional formula. As mentioned earlier the calculus **SBL** is a predicate calculus without function symbols and with '∈' as the only predicate symbol. Such a calculus is undecidable in general (see for instance [3]). The translation procedure described above produces, however, formulas lying in a certain subset of **SBL** which we call **SBL**^{2}. The variables of a formula of this subset can always be divided into one set of variables occurring only on the left-hand side of the '∈' predicate (element variables) and one set of variables occurring only on the right-hand side of the '∈' predicate (class variables). Furthermore only quantifiers for element variables are introduced in the course of the translation from **SL** to **SBL** because due to the syntax of **SL** the first formula of (1) is not applied. A formula in **SBL**^{2} can be transformed to a propositional formula $\mathcal{F}(\rho_1, \ldots, \rho_m)$ where $\rho_1, \ldots, \rho_m$ are **SBL**^{2}-formulas of the form:

$$\underbrace{\forall x(x{\in}A_1{\vee}\ldots{\vee}x{\in}A_n)}_{\rho_1},\ \underbrace{\forall x(x{\in}A_1{\vee}\ldots{\vee}\neg(x{\in}A_n))}_{\rho_2},\ \ldots,\ \underbrace{\forall x(\neg(x{\in}A_1){\vee}\ldots{\vee}\neg(x{\in}A_n))}_{\rho_m}$$

This is possible because due to the restricted quantification a universal quantifier $\forall x$ can be moved (after elimination of $\Rightarrow$ and $\Leftrightarrow$) to the right until a disjunction of the form $(x \in A_1 \vee \ldots \vee x \in A_k)$ is encountered (where $k \leq n$, and atomic formulas are possibly negated). This can be achieved by the use of the formulas below where $\Phi_{(x)}, \Psi_{(x)}$ denote formulas containing x as a free variable and Φ, Ψ are formulas *not* containing x as a free variable.

$$\underset{\text{SBL}}{\vdash} \forall x(\Phi_{(x)} \wedge \Psi_{(x)}) \Leftrightarrow ((\forall x\ \Phi_{(x)}) \wedge (\forall x\ \Psi_{(x)})),$$
$$\underset{\text{SBL}}{\vdash} \forall x(\Phi \vee \Psi_{(x)}) \Leftrightarrow (\Phi \vee (\forall x\ \Psi_{(x)})), \quad \underset{\text{SBL}}{\vdash} \Phi \Leftrightarrow (\Phi \vee \Psi) \wedge (\Phi \vee \neg \Psi) \tag{4}$$

In Example 1 below we demonstrate the complete transformation of an **SL**-formula to a propositional formula. We use the abbreviations of (3), the second formula of (1), the formulas of (4), and the notation:
$P_x^A := x \in A, \quad \overline{P_x^A} := \neg(x \in A), \quad [P_x^A, \ldots, P_x^C] := P_x^A \vee \ldots \vee P_x^C.$

Example 1

$A \cap B \subseteq C \Rightarrow (A \subseteq C \vee B \subseteq C)$

$\{v | v \in A \wedge v \in B\} \subseteq C \Rightarrow (A \subseteq C \vee B \subseteq C)$

$(\forall x(x \in \{v | v \in A \wedge v \in B\} \Rightarrow x \in C)) \Rightarrow ((\forall y(y \in A \Rightarrow y \in C)) \vee (\forall z(z \in B \Rightarrow z \in C)))$

$(\forall x((x \in A \wedge x \in B) \Rightarrow x \in C)) \Rightarrow ((\forall y(y \in A \Rightarrow y \in C)) \vee (\forall z(z \in B \Rightarrow z \in C)))$

$(\forall x((P_x^A \wedge P_x^B) \Rightarrow P_x^C)) \Rightarrow ((\forall y(P_y^A \Rightarrow P_y^C)) \vee (\forall z(P_z^B \Rightarrow P_z^C)))$

$\neg(\forall x(\neg(P_x^A \wedge P_x^B) \vee P_x^C)) \vee ((\forall y(\overline{P_y^A} \vee P_y^C)) \vee (\forall z(\overline{P_z^B} \vee P_z^C)))$

$\neg(\forall x[\overline{P_x^A}, \overline{P_x^B}, P_x^C]) \vee (\forall y[\overline{P_y^A}, P_y^C]) \vee (\forall z[\overline{P_z^B}, P_z^C])$

$\neg(\forall x[\overline{P_x^A}, \overline{P_x^B}, P_x^C]) \vee (\forall y([\overline{P_y^A}, P_y^C, P_y^B] \wedge [\overline{P_y^A}, P_y^C, \overline{P_y^B}])) \vee (\forall z([\overline{P_z^B}, P_z^C, P_z^A] \wedge [\overline{P_z^B}, P_z^C, \overline{P_z^A}]))$

$\neg(\forall x[\overline{P_x^A}, \overline{P_x^B}, P_x^C]) \vee (\forall y[\overline{P_y^A}, P_y^C, P_y^B] \wedge \forall y[\overline{P_y^A}, P_y^C, \overline{P_y^B}]) \vee (\forall z[\overline{P_z^B}, P_z^C, P_z^A] \wedge \forall z[\overline{P_z^B}, P_z^C, \overline{P_z^A}])$

$\neg(\underbrace{\forall x[\overline{P_x^A}, \overline{P_x^B}, P_x^C]}_{\rho_7}) \vee (\underbrace{\forall x[\overline{P_x^A}, P_x^B, P_x^C]}_{\rho_5} \wedge \underbrace{\forall x[\overline{P_x^A}, \overline{P_x^B}, P_x^C]}_{\rho_7}) \vee (\underbrace{\forall x[P_x^A, \overline{P_x^B}, P_x^C]}_{\rho_3} \wedge \underbrace{\forall x[\overline{P_x^A}, \overline{P_x^B}, P_x^C]}_{\rho_7})$

The resulting propositional formula is $\mathcal{F} = \neg\rho_7 \vee (\rho_5 \wedge \rho_7) \vee (\rho_3 \wedge \rho_7)$. In general the formula can have $m = 2^n$ ρ-arguments if n is the number of different variables occurring in the **SL**-formula. The ρ-arguments are independent of one another except that all of them cannot be true simultaneously. The validity of a formula in $\mathbf{SBL}^2$ can be decided by a Boolean valuation of the propositional formula $\mathcal{F}$ excluding the case that all ρ-arguments are simultaneously true. Note at this point that a formula $\mathcal{F}$ in which all possible ρ-arguments occur represents the worst case. In the great majority of cases only a small subset of the theoretically possible ρ-arguments actually occur in $\mathcal{F}$ making the evaluation of $\mathcal{F}$ much more efficient (see Example 1). In the case of a satisfiable **SL**-formula it is possible to construct a collection of finite sets from a Boolean valuation $b : \{\rho_1, \ldots, \rho_m\} \rightarrow \{false, true\}^m$ of the corresponding formula $\mathcal{F}$. These sets can subsequently serve to generate Venn diagrams as counterexamples (or examples) of the **SL**-formula. A description of such a generation can be found on the homepage of the tool.

The implementation. The Hilberticus tool decides whether an **SL**-formula is true, satisfiable, or inconsistent in the model $\mathcal{M}$ described in Section 2. Given an **SL**-formula the tool generates a strictly typed abstract syntax tree (AST) using the syntax and the priorities of functions and logical connectives described in Section 2. Then the tree is transformed to an AST representing the corresponding **SBL**-formula which is subsequently decided according to the described procedure. The tool is written in Java© to be easily accessible. It is made up of different modules which are tested and bound together within the Electronic Tool Integration platform (ETI) [4], the experimental platform of the *Int. J. STTT.* The integrated parser was generated using SableCC, a suitable Java© compiler compiler [5]. The ASTs of **SL**- and **SBL**-formulas can be visualized using the PLGraph class library which is supplied by the ETI platform.

4 Related and future work

The language **SL** is a sublanguage of a language first described in [6] and later named Multi-level Syllogistic (**MLS**), see for instance [7]. Later on, various variants of (**MLS**) have been shown to be decidable. The most recent decision procedures use semantic tableaux [7]. The semantics of the languages is based on Zermelo-Fraenkel set theory or parts of it. The sublanguage of **MLS** which is the most similar to **SL** is called **2LS** and is described together with a decision procedure in [8]. As mentioned earlier the Hilberticus tool is the first implementation of the decision procedure described in Section 3. It was chosen because it supplies a convenient way to obtain finite sets for Venn diagram generation and because of the calculus $\mathbf{L}\boldsymbol{\in}$ offering a natural possibility to extend the language. The use of abstraction terms makes it an easy task to introduce new function symbols. With a test implementation containing a generalized decision procedure we were able to find the incorrect formula $\bigcup(M \cap N) = \bigcup M \cap \bigcup N$ in [9, p.545], a book which is used as reference for all kinds of mathematical formulas. We are currently working on a version of the tool which uses tableaux based procedures such as mentioned above and the described translation to **SBL** in combination with decision (or verification) procedures for predicate and monadic logic. In this context the ETI platform supplies the ideal environment for the integration, comparison, and testing of these translation and decision procedures.

References

1. Glubrecht, J.-M., Oberschelp, A., Todt, G.: Klassenlogik. BI Wissenschaftsv. (1983)
2. Oberschelp, A.: Allgemeine Mengenlehre. BI Wissenschaftsverlag (1994)
3. Enderton, H.B.: A mathematical introduction to logic. Acad. Press (1972)
4. Steffen, B., Margaria, T., Braun, V.: The Electronic Tool Integration platform: concepts and design. Int. J. STTT (1997) 1:9-30, eti.cs.uni-dortmund.de
5. Gagnon, E. M., Hendren, L. J.: SableCC. In Proc. TOOLS'98, IEEE
6. Ferro, A., Omodeo, E., Schwartz, J.T.: Decision Procedures for Elementary Sublanguages of Set Theory I. Comm. Pure Appl. Math. Vol. XXXIII (1980) 599–608
7. Cantone, D., Ferro, A.: Techniques of Computable Set Theory with Applications to Proof Verification. Comm. Pure Appl. Math. Vol. XLVIII (1995) 901–945
8. Cantone, D., Omodeo, E., Policriti, A.: Set Theory for Computing. Springer (2001)
9. Bronstein, I. N., Semendjajew, K. A.: Taschenbuch der Mathematik. (25. Aufl., 1991)