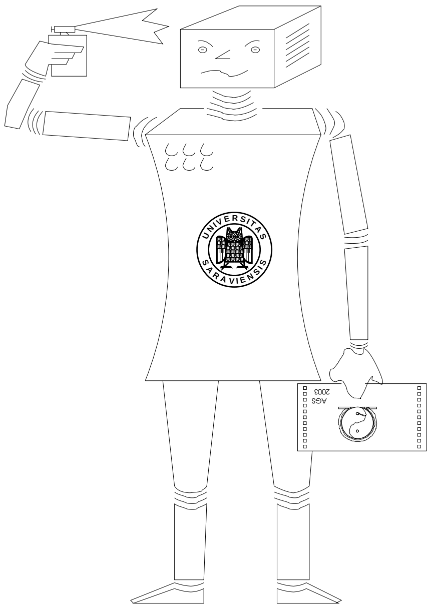
SEKI-REPORT ISSN 1437-4447

UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D–66123 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/



Disproving False Conjectures

Serge Autexier
FR 6.2 Informatik, Saarland University & German
Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany, autexier@dfki.de

Carsten Schürmann
Yale University, New Haven, USA
carsten@cs.yale.edu

SEKI Report SR–2003–06

The following publication is a short version of this SEKI Report:
Serge Autexier, Carsten Schürmann. *Disproving False Conjectures.* Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning; Almaty, Kazakhstan.

The above short version of this SEKI Report successfully passed the following external anonymous referee process:
LPAR 2003, 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning; full paper, 15 pp.

Editor of SEKI series:
Claus-Peter Wirth
FR Informatik, Universität des Saarlandes, D–66123 Saarbrücken, Germany
E-mail: `cp@ags.uni-sb.de`
WWW: `http://www.ags.uni-sb.de/~cp/welcome.html`

Disproving False Conjectures

Serge Autexier
FR 6.2 Informatik, Saarland University & German
Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany, autexier@dfki.de

Carsten Schürmann
Yale University, New Haven, USA
carsten@cs.yale.edu

July 22, 2003

Abstract

For automatic theorem provers it is as important to disprove false conjectures as it is to prove true ones, especially if it is not known ahead of time if a formula is derivable inside a particular inference system. Situations of this kind occur frequently in inductive theorem proving systems where failure is a common mode of operation. This paper describes an abstraction mechanism for first-order logic over an arbitrary but fixed term algebra to second-order monadic logic with 0 successor functions. The decidability of second-order monadic logic together with our notion of abstraction yields an elegant criterion that characterizes a subclass of unprovable conjectures.

The research on automated theorem proving is inspired by Leibniz' dream to develop a "lingua characteristica" together with a "calculus ratiocinator" in order to mechanize logical reasoning. He advocated using the purely rational and incorruptible mechanized reasoners in order to decide whether a given logical consequence holds or not. While this vision has spurred a lot of research devoted to proving *true* conjectures, disproving *false* conjectures that occur frequently in proof assistants, inductive theorem provers, and logical programming systems has attracted far less interest.

In fact, in most theorem proving systems, the default mode of operation is failure: conjectures are often entered in the hope that they are correct, the proof by cases is typically triggered by failure to prove a given subgoal, and even in logic programming, backtracking is always preceded by failure to construct a proof of a subgoal be it in Horn logic or hereditary Harrop logic [8].

In practice, the development of conjectures is an evolutionary process and typically a true conjecture is the result of a sequence of false conjectures and their disproofs. Thus, research on automatic disproving of false conjectures is equally important as automatic proving of true conjectures. Automatic disproving is of increasing relevance in the context of formal software development [2, 3], where early detection of flaws in programs reduces the overall development cost.

The key idea underlying our technique presented in this paper consists of the definition of a representational abstraction function of first-order logic formulas into a decidable fragment of second-order logic, namely second-order monadic logic without successor functions, S0S [9]. The abstraction function is effectively polynomial-time computable, preserves the structural form of the original formula, and most importantly preserves non-provability. Second-order monadic logic is decidable, and therefore disproving a conjecture in S0S implies contrapositively that the original conjecture could not have been provable either. The decision procedure of S0S is PSPACE-complete [13], but will always report true or false. Only if no proof in S0S exists, we can be sure that the conjecture is indeed false. If a proof exists, in terms of provability, nothing can be learned from it. However, the proof may contain vital information that can assist the theorem prover to try to prove the conjecture, a question which we will consider in future work.

In preliminary studies [10], we have developed a similar criterion for the first-order meta-logic $\mathcal{M}_\omega^+$ for the logical framework LF [5], although without negation, disjunction, or implication. Besides truth and falsehood we did not consider any other logical constants or predicates. The abstraction mechanism presented in this paper, on the other hand, scales to first-order logic with equational theories (e.g. Peano Arithmetic) and is based on Leibniz equality which is prevalent in many higher-order theorem proving systems [1, 12].

The paper is organized as follows: In Sec. 1 we define syntax and sequent calculi of first-order and second-order monadic logics. The abstraction of first-order logic formulas to second-order logic formulas and the relevant meta theory is presented in Sec. 2. In Sec. 3 we extend our techniques to first-order logic with equality and show in Sec. 4 that the class of disprovable formulas includes formulas with infinite counter-models. In Sec. 5 we present details about the implementation of the technique before concluding and assessing results in Sec. 6.

1 First-Order Logic and Second-Order Monadic Logic

We recapitulate the definitions of first-order and second-order monadic logic with 0 successors (S0S) as well as the decidability result of second-order modal logic [9] that is relevant for the technique presented in this paper.

1.1 First-Order Logic

Definition 1.1 (First-Order Logic Formulas) Let $\mathcal{T}(\mathcal{C}, \mathcal{V})$ be a term algebra freely generated from a set of constant symbols $\mathcal{C}$ and a list of pairwise different variable symbols $\mathcal{V}$. Let $\mathcal{P}$ be a set of predicates. Then first-order logic formulas are defined by

$$\text{First-order Logic Formulas:} \quad F \quad ::= \quad P(t_1 \ldots t_n) \mid \top \mid \bot \mid F_1 \supset F_2 \mid F_1 \wedge F_2 \mid \neg F \mid \forall x.\, F \mid \exists x.\, F$$

$$\begin{array}{ll}\textbf{Terms:} & VC(x) := [x] \;\; VC(c) = [c] \;\; VC(f(t_1,\ldots,t_n)) := \bigoplus_{i=1}^{n} VC(t_i) \\ \textbf{Formulas:} & VC(P(t_1,\ldots,t_n)) := \bigoplus_{i=1}^{n} VC(t_i) \;\; VC(\top) = VC(\bot) := [] \\ & VC(F_1 \supset F_2) = VC(F_1 \wedge F_2) := VC(F_1) \oplus VC(F_2) \\ & VC(\neg F) := VC(F) \;\; VC(\forall x.\, F) = VC(\exists x.\, F) := VC(F) \setminus \{x\}\end{array}$$

Figure 1: List of constants and free variables in formulas and terms, where $[x]$ denotes the singleton list with the variable x, $[c]$ the singleton list with the constant c, $\oplus$ denotes the concatenation of lists, and $L \setminus \{x\}$ denotes the list obtained from L by removing any occurrence of the variable x.

where $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ and $P \in \mathcal{P}$. In first-order logic, we write x, y, z for variables. For formulas F and terms t we write $VC(F)$ and $VC(t)$ to refer to the list[1] of free variables and constants in F and t (cf. Fig. 1).

Substitutions are capture avoiding and play an important role in this paper, especially in the proof of the soundness Theorem 2.7. We do not distinguish between substitutions for first-order or second-order monadic logic.

Definition 1.2 (Substitutions) A substitution σ is a syntactically defined object $\sigma ::= \cdot \mid \sigma, t/x$. As usual, we write $\sigma(x)$ to apply σ to the variable x and the domain of a substitution is the set of variables for which $\sigma(x)$ is defined. The domain of a substitution is always finite.

Definition 1.3 (First-Order Substitution Application) We denote by $[\sigma]t$ and $[\sigma]F$ the standard application of σ to first-order terms and formulas.

A sequent calculus for classical first-order logic is given in Fig. 2. All rules are standard. The subscript $_1$ in the rule names identifies the quantifier rules as first-order. The superscript ^{a} indicates that a is fresh in $\Gamma \Longrightarrow \forall x.\, F$ for $\forall_1 \mathsf{I}^a$ and in $\Gamma \Longrightarrow H$ for $\exists_1 \mathsf{E}^a$. First-order logic provides a foundation of several theorem proving systems, Spass, INKA, and others, and we illustrate its use with our running example about binary trees.

Example 1.4 (Binary trees) In first-order logic, properties of trees and paths can be expressed as formulas ranging over terms generated by a term algebra that consists of two constant symbols here (for the empty path) and leaf (for leaves in a tree), two unary function symbols left and right (for paths denoting respectively left and right subtrees), and one binary function symbol node (for non-leaf nodes of trees). We use the $\mathsf{validtree}$ and $\mathsf{validpath}$ as unary predicates that describe the well-formedness of trees and paths, respectively, mirror and $\mathsf{reflect}$ as binary predicates, where $\mathsf{mirror}(t, t')$ stands for t' is a tree that is derived from t by subtreewise exchanging left and right subtrees, and $\mathsf{reflect}(p, p')$ for p' is a path that is derived from p by exchanging constant left by right and vice versa. A set of axioms that relate terms is given in Fig. 3.

[1]We define $VC(t)$ as a list of variables and constants as the order of the symbols simplifies the proofs. However, the reader may think of $VC(t)$ as a set.

$$\frac{}{\Gamma, F \Longrightarrow \Delta, F}\ \mathsf{ax} \qquad \frac{}{\Gamma \Longrightarrow \Delta, \top}\ \top\mathsf{R} \qquad \frac{}{\Gamma, \bot \Longrightarrow \Delta}\ \bot\mathsf{L}$$

$$\frac{\Gamma \Longrightarrow \Delta}{\Gamma, F \Longrightarrow \Delta}\ \mathsf{weak\ L} \qquad \frac{\Gamma \Longrightarrow \Delta}{\Gamma \Longrightarrow F, \Delta}\ \mathsf{weak\ R}$$

$$\frac{\Gamma \Longrightarrow \Delta, F \quad \Gamma \Longrightarrow \Delta, G}{\Gamma \Longrightarrow \Delta, F \wedge G}\ \wedge\mathsf{R} \qquad \frac{\Gamma, F, G, \Longrightarrow \Delta}{\Gamma, F \wedge G \Longrightarrow \Delta}\ \wedge\mathsf{L}$$

$$\frac{\Gamma \Longrightarrow \Delta, F, G}{\Gamma \Longrightarrow \Delta, F \vee G}\ \vee\mathsf{R} \qquad \frac{\Gamma, F \Longrightarrow \Delta \quad \Gamma, G \Longrightarrow \Delta}{\Gamma, F \vee G \Longrightarrow \Delta}\ \vee\mathsf{L}$$

$$\frac{\Gamma, F \Longrightarrow \Delta, G}{\Gamma \Longrightarrow \Delta, F \supset G}\ \supset\mathsf{R} \qquad \frac{\Gamma, \Longrightarrow \Delta, F \quad \Gamma, G \Longrightarrow \Delta}{\Gamma, F \supset G \Longrightarrow \Delta}\ \supset\mathsf{L}$$

$$\frac{\Gamma, F \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta, \neg F}\ \neg\mathsf{R} \qquad \frac{\Gamma \Longrightarrow \Delta, F}{\Gamma, \neg F \Longrightarrow \Delta}\ \neg\mathsf{L}$$

$$\frac{\Gamma \Longrightarrow \Delta, [a/x]F}{\Gamma \Longrightarrow \Delta, \forall x.\, F}\ \forall_1\mathsf{R}^a \qquad \frac{\Gamma, \forall x.\, F, [t/x]F \Longrightarrow \Delta}{\Gamma, \forall x.\, F \Longrightarrow \Delta}\ \forall_1\mathsf{L}$$

$$\frac{\Gamma \Longrightarrow \Delta, \exists x.\, F, [t/x]F}{\Gamma \Longrightarrow \Delta, \exists x.\, F}\ \exists_1\mathsf{R} \qquad \frac{\Gamma, [a/x]F \Longrightarrow \Delta}{\Gamma, \exists x.\, F \Longrightarrow \Delta}\ \exists_1\mathsf{L}^a$$

$$\frac{\Gamma, F \Longrightarrow \Delta \quad \Gamma \Longrightarrow F, \Delta}{\Gamma \Longrightarrow \Delta}\ \mathsf{Cut}(F)$$

Figure 2: Sequent Calculus for Classical First-Order Logic

A property about binary trees that one may be interested in is to show that mirrored subtrees are preserved under reflecting paths which can be formally expressed as

$$\begin{array}{l}\forall t.\, \forall s.\, \forall p.\, (\mathsf{validtree}(t) \wedge \mathsf{validtree}(s) \wedge \mathsf{validpath}(p) \wedge \mathsf{subtree}(t, p, s)) \\ \quad \supset \exists t'.\, \exists s'.\, \exists p'.\, (\mathsf{validtree}(t') \wedge \mathsf{validtree}(s') \wedge \mathsf{validpath}(p') \wedge \mathsf{subtree}(t', p', s') \\ \quad \wedge\ \mathsf{mirror}(t, t') \wedge \mathsf{reflect}(p, p') \wedge \mathsf{mirror}(s, s')\end{array}$$

Without induction principles, this theorem is not provable in first-order logic. □

1.2 Second-Order Monadic Logic without Successor Functions

Second-order monadic logic without successor functions (S0S) restricts atomic formulas to the form $P(x)$ or $X(x)$ where $x \in \mathcal{V} \cup \mathcal{C}$ is either a variable or a constant, P is a unary predicate, and X is a unary variable that ranges over unary predicates.

Definition 1.5 (Second-Order Logic Formulas S0S) Let $\mathcal{T}(\mathcal{C}, \mathcal{V})$ be a term algebra with constants and variables only, and $\mathcal{P}$ be defined as above in Definition 1.1 and $\mathcal{W}$ a list of pairwise distinct second-order variable names. Second-order monadic logic formulas are defined by

$$\begin{array}{lcl}\text{S0S formulas:}\ G & ::= & P(x) \mid P(c) \mid X(x) \mid X(c) \mid \top \mid \bot \mid G_1 \supset G_2 \mid G_1 \wedge G_2 \\ & & \mid \neg G \mid \forall x.\, G \mid \exists x.\, G \mid \forall X.\, G \mid \exists X.\, G\end{array}$$

$$\begin{array}{l}
\mathsf{validtree}(\mathsf{leaf}) \\
\forall t_1.\, \forall t_2.\, \mathsf{validtree}(t_1) \wedge \mathsf{validtree}(t_2) \supset \mathsf{validtree}(\mathsf{node}(t_1, t_2)) \\
\mathsf{validpath}(\mathsf{here}) \\
\forall p.\, \mathsf{validpath}(p) \supset \mathsf{validpath}(\mathsf{left}(p)) \\
\forall p.\, \mathsf{validpath}(p) \supset \mathsf{validpath}(\mathsf{right}(p)) \\
\mathsf{mirror}(\mathsf{leaf}, \mathsf{leaf}) \\
\forall t_1.\, \forall t_1'.\, \forall t_2.\, \forall t_2'.\, \mathsf{mirror}(t_1, t_1') \wedge \mathsf{mirror}(t_2, t_2') \supset \mathsf{mirror}(\mathsf{node}(t_1, t_2), \mathsf{node}(t_2', t_1')) \\
\mathsf{reflect}(\mathsf{here}, \mathsf{here}) \\
\forall p.\, \forall p'.\, \mathsf{reflect}(p, p') \supset \mathsf{reflect}(\mathsf{left}(p), \mathsf{right}(p')) \\
\forall p.\, \forall p'.\, \mathsf{reflect}(p, p') \supset \mathsf{reflect}(\mathsf{right}(p), \mathsf{left}(p')) \\
\forall t.\, \mathsf{subtree}(t, \mathsf{here}, t) \\
\forall t_1.\, \forall t_2.\, \forall p.\, \forall t'.\, \mathsf{subtree}(t_1, p, t') \supset \mathsf{subtree}(\mathsf{node}(t_1, t_2), \mathsf{left}(p), t') \\
\forall t_1.\, \forall t_2.\, \forall p.\, \forall t'.\, \mathsf{subtree}(t_2, p, t') \supset \mathsf{subtree}(\mathsf{node}(t_1, t_2), \mathsf{right}(p), t')
\end{array}$$

Figure 3: Sample set of axioms defining properties of trees

$$\frac{\Gamma \Longrightarrow [p/X]A, \Delta}{\Gamma \Longrightarrow \forall X.\, A, \Delta}\, \forall \mathsf{R}^p \qquad \frac{\Gamma, \forall X.\, A, [P/X]A \Longrightarrow \Delta}{\Gamma, \forall X.\, A \Longrightarrow \Delta}\, \forall \mathsf{L}$$

$$\frac{\Gamma \Longrightarrow [P/X]A, \exists X.\, A, \Delta}{\Gamma \Longrightarrow \exists X.\, A, \Delta}\, \exists \mathsf{R} \qquad \frac{\Gamma, [p/x]A \Longrightarrow \Delta}{\Gamma, \exists x.\, A \Longrightarrow \Delta}\, \exists \mathsf{L}^p$$

Figure 4: Additional Rules for second-order logic

where $x \in \mathcal{V}$, $c \in \mathcal{C}$, $X \in \mathcal{W}$ and $P \in \mathcal{P}$. In second-order monadic logic, we write x, y, z for variables, and X, Y, Z for variables that range over predicates.

The sequent calculus for classical S0S is obtained by adding four left and right rules for the second-order quantifiers to the respective first-order natural deduction calculi as depicted in Fig. 4 where P is any predicate from $\mathcal{P}$ and p is new with respect to the sequent. Since we consider second-order monadic logic without successors, $t \in \mathcal{V} \cup \mathcal{C}$ in rules $\exists_1\mathsf{I}$ and $\forall_1\mathsf{E}$, respectively. For the purpose of our paper the main result about second-order monadic logic is that it is decidable, which has been proved by Rabin [9].

Theorem 1.6 (Rabin, 1969) *Second-order monadic logic with k successor functions is decidable.* $\square$

2 Abstraction

It is well-known that brute force search for proofs of conjectures may easily exhaust system resources regarding space and time. If a conjecture is true, the traversal of the search space in one way or another is necessary to find the derivation that is known to exist. Often, however, interim conjectures are not necessarily known to be derivable. These situations arise frequently in systems where induction principles are not axiomatized but encoded via special elimination rules. In many inductive theorem provers, therefore, failure to find a derivation in the non-inductive fragment indicates that subsequent case analyses are necessary and failure is therefore the predominant way of operation.

Of course, before a theorem prover can meaningfully fail, it must have visited every node in the search space that is potentially infinite. Alternatively, following the algorithm outlined in this paper, it is often possible to disprove formally a conjecture. Our proposed technique relies on an abstraction into second-order monadic logic without successor functions that is known to be decidable. If the abstracted formula is false, by the soundness of abstraction (Theorem 2.7), the original formula is false as well. Therefore, following the proposed classifications of abstractions by Giunchiglia and Walsh [4][2], our notion of abstraction satisfies the properties of a TI abstraction with a consistent abstract space. For the domain of first-order logic, first-order monadic logic would suffice as abstract space, but equality (see Sec. 3) requires the use of second-order monadic logic.

The abstraction can be intuitively explained as follows. A derivation $\cdot \Longrightarrow P(t_1, \ldots, t_n)$ must contain information about the individual t_i's in one form or another. Without axiomatizing this relation, we instead propose to approximate it, and we rewrite $P(t_1, \ldots, t_n)$ to a conjunction of unary atomic formulas $P(x)$ and $P(c)$ for any variable x and any constant c that occurs in the terms. The abstraction preserves the structure of a formula, and is defined as follows.

Definition 2.1 (Abstraction)

$$
\begin{array}{rclr@{\qquad}rclr}
\alpha(\top) & := & \top & (1) & \alpha(F_1 \supset F_2) & := & \alpha(F_1) \supset \alpha(F_2) & (5)\\
\alpha(\bot) & := & \bot & (2) & \alpha(\neg F) & := & \neg(\alpha(F)) & (6)\\
\alpha(F_1 \vee F_2) & := & \alpha(F_1) \vee \alpha(F_2) & (3) & \alpha(\forall x.\, F) & := & \forall x.\, \alpha(F) & (7)\\
\alpha(F_1 \wedge F_2) & := & \alpha(F_1) \wedge \alpha(F_2) & (4) & \alpha(\exists x.\, F) & := & \exists x.\, \alpha(F) & (8)
\end{array}
$$

$$\alpha(P(t_1, \ldots, t_n)) := \bigwedge_{x \in VC(P(t_1,\ldots,t_n))} P(x) \qquad (9)$$

The cases (1)–(8) are straightforward, which leaves (9) to be explained. In case (9) the expresion $\bigwedge_{x \in VC(P(t_1,\ldots,t_n))} P(x)$ is the conjunction of formulas defined by

$$\bigwedge_{x \in []} P(x) := \top, \quad \bigwedge_{x \in [x']} P(x) := P(x'), \quad \text{and} \quad \bigwedge_{x \in [x'] \oplus L} P(x) := P(x') \wedge \left(\bigwedge_{x \in L} P(x) \right)$$

Example 2.2 We illustrate the technique by abstracting the axioms depicted in Fig. 3. The result is shown in Fig. 5.

The following lemma ensures that the abstraction of any first-order logic formula is always a second-order monadic formula with respect to S0S.

[2]This paper also provides an overview of different abstraction mechanisms.

$$\begin{array}{l}
\mathsf{validtree}(\mathsf{leaf})\\
\forall t_1.\, \forall t_2.\, \mathsf{validtree}(t_1) \wedge \mathsf{validtree}(t_2) \supset \mathsf{validtree}(t_1) \wedge \mathsf{validtree}(t_2)\\
\mathsf{validpath}(\mathsf{here})\\
\forall p.\, \mathsf{validpath}(p) \supset \mathsf{validpath}(p)\\
\forall p.\, \mathsf{validpath}(p) \supset \mathsf{validpath}(p)\\
\mathsf{mirror}(\mathsf{leaf}) \wedge \mathsf{mirror}(\mathsf{leaf})\\
\forall t_1.\, \forall t_1'.\, \forall t_2.\, \forall t_2'.\, \mathsf{mirror}(t_1) \wedge \mathsf{mirror}(t_1') \wedge \mathsf{mirror}(t_2) \wedge \mathsf{mirror}(t_2')\\
\quad \supset \mathsf{mirror}(t_1) \wedge \mathsf{mirror}(t_2) \wedge \mathsf{mirror}(t_2') \wedge \mathsf{mirror}(t_1')\\
\mathsf{reflect}(\mathsf{here}) \wedge \mathsf{reflect}(\mathsf{here})\\
\forall p.\, \forall p'.\, \mathsf{reflect}(p) \wedge \mathsf{reflect}(p') \supset \mathsf{reflect}(p) \wedge \mathsf{reflect}(p')\\
\forall p.\, \forall p'.\, \mathsf{reflect}(p) \wedge \mathsf{reflect}(p') \supset \mathsf{reflect}(p) \wedge \mathsf{reflect}(p')\\
\forall t.\, \mathsf{subtree}(t) \wedge \mathsf{subtree}(\mathsf{here}) \wedge \mathsf{subtree}(t)\\
\forall t_1.\, \forall t_2.\, \forall p.\, \forall t'.\, \mathsf{subtree}(t_1) \wedge \mathsf{subtree}(p) \wedge \mathsf{subtree}(t') \supset\\
\quad \mathsf{subtree}(t_1) \wedge \mathsf{subtree}(t_2) \wedge \mathsf{subtree}(p) \wedge \mathsf{subtree}(t')\\
\forall t_1.\, \forall t_2.\, \forall p.\, \forall t'.\, \mathsf{subtree}(t_2) \wedge \mathsf{subtree}(p) \wedge \mathsf{subtree}(t') \supset\\
\quad \mathsf{subtree}(t_1) \wedge \mathsf{subtree}(t_2) \wedge \mathsf{subtree}(p) \wedge \mathsf{subtree}(t')
\end{array}$$

Figure 5: Abstractions of the sample set of axioms

Lemma 2.3 *For any first-order logic formula F, $\alpha(F)$ is a second-order monadic formula without successor functions, and it holds $VC(F) = VC(\alpha(F))$.*

Proof. *The proof is by induction over the structure of the formula F:*

Base Case: *If F is an atomic formula, then F is either $\top$, or $\bot$, or of the form $P(t_1, \ldots, t_n)$. If $F \in \{\top, \bot\}$, then $\alpha(F) = F$ and thus $\alpha(F)$ is a second-order monadic logic formula and it holds $VC(F) = VC(\alpha(F))$.*

Otherwise, if $F := P(t_1, \ldots, t_n)$, then $\alpha(P(t_1, \ldots, t_n))$ is

$$\bigwedge_{x \in VC(P(t_1,\ldots,t_n))} P(x).$$

By simple induction over the length of $VC(P(t_1, \ldots, t_n))$ it follows that this formula is indeed a second-order monadic logic formula without successor functions. Furthermore $VC(P(t_1, \ldots, t_n)) = VC(\bigwedge_{x \in \bigoplus_{i=1}^{n} VC(t_i)} P(x))$ follows directly from the definition.

Induction Step: *Assume that the abstractions of F_1, F_2 are second-order monadic logic formulas and it holds $VC(F_i) = VC(\alpha(F_i))$, $i = 1, 2$. Then for all $\circ \in \{\wedge, \vee, \supset\}$ obviously $\alpha(F_1 \circ F_2) := \alpha(F_1) \circ \alpha(F_2)$ is a second-order monadic logic formula. The analogous argument applies to $\neg F_1$, $\forall x.\, F_1$, and $\exists x.\, F_1$.*

For the second part of the lemma it holds for $F_1 \circ F_2$:

$$\begin{aligned}
VC(F_1 \circ F_2) &= VC(F_1) \oplus VC(F_2) \stackrel{I.H.}{=} \alpha(VC(F_1)) \oplus \alpha(VC(F_2))\\
&= \alpha(VC(F_1)) \circ \alpha(VC(F_2))
\end{aligned}$$

and analogously for $\neg F_1$. *For* $\mathsf{Q} \in \{\forall, \exists\}$ *it holds*

$$VC(\mathsf{Q}x.\,F_1) = VC(F_1) \setminus \{x\} \stackrel{I.H.}{=} VC(\alpha(F_1)) \setminus \{x\} = VC(\mathsf{Q}x.\,\alpha(F_1))$$

□

We now address the question of how substitutions and abstraction interact. Following Definition 1.2 the standard definition of substitutions may contain non-monadic terms, which complicates the interaction with abstraction. Consider the following example. Let $P(f(x, y))$ be a predicate and $\sigma = g(u, v)/x$ a substitution. Applying σ naively to the result of abstraction $P(x) \wedge P(y)$ would yield $P(g(u, v)) \wedge P(y)$, which is not an S0S formula and differs from

$$\alpha([\sigma](P(f(x, y)))) = \alpha(P(f(g(u, v), y))) = P(u) \wedge P(v) \wedge P(y).$$

Thus, substitution application of σ to t differs from the standard form of application, since it is required to flatten the structure of atomic formulas, as well. It is defined over the structure of t and σ, simultaneously.

Definition 2.4 (Flattening substitution application) We denote by $[\![\sigma]\!](t)$ and $[\![\sigma]\!](F)$ the application of the homomorphic extension of σ to second-order terms and formulas defined by:

$$\begin{array}{rrcll}
 & [\![\sigma]\!](P(x)) & := & \bigwedge\limits_{y \in VC(\sigma(x))} P(y) & (10)\\
 & [\![\sigma]\!](\neg F) & := & \neg([\![\sigma]\!](F)) & (11)\\
\text{for } \circ \in \{\wedge, \vee, \supset\} & [\![\sigma]\!](F_1 \circ F_2) & := & [\![\sigma]\!](F_1) \circ [\![\sigma]\!](F_2) & (12)\\
\text{for } \mathsf{Q} \in \{\forall, \exists\} & [\![\sigma]\!](\mathsf{Q}x.\,F) & := & \mathsf{Q}x.\,[\![\sigma, x/x]\!](F) & (13)
\end{array}$$

where $(\sigma, x/x)$ denotes the substitution that maps x to x and otherwise is identical to σ.

Substitutivity in first-order logic and S0S commute with abstraction, which is the crucial property used at several occasions in the proof of the soundness Theorem 2.7.

Lemma 2.5 *Let F be a first-order logic formula and σ a first-order substitution. Then it holds:*

$$\alpha([\sigma]F) = [\![\sigma]\!](\alpha(F))$$

Proof. *by induction on the structure of F. We only show two representative cases.*

Case: *If F is an atomic formula of the form $P(t_1, \ldots, t_n)$ then*

$$\begin{array}{rcl}
\alpha([\sigma]P(t_1, \ldots, t_n)) & = & \alpha(P([\sigma]t_1, \ldots, [\sigma]t_n))\\
 & = & \bigwedge\limits_{x \in \bigoplus_{i=1}^{n} VC([\sigma]t_i)} P(x)
\end{array}$$

$$
\begin{aligned}
&= \bigwedge_{y \in \bigoplus_{i=1}^{n} VC(t_i)} \left(\bigwedge_{x \in VC(\sigma(y))} P(x) \right) \\
&= \bigwedge_{y \in \bigoplus_{i=1}^{n} VC(t_i)} [\![\sigma]\!] P(y) \\
&= [\![\sigma]\!] \left(\bigwedge_{y \in \bigoplus_{i=1}^{n} VC(t_i)} P(y) \right) \\
&= [\![\sigma]\!](\alpha(P(t_1, \ldots, t_n)))
\end{aligned}
$$

Case: $F = \mathsf{Q}x\,.F'$

$$
\begin{aligned}
\alpha([\sigma](\mathsf{Q}x\,.F')) &= \alpha(\mathsf{Q}x\,.[\sigma, x/x](F')) \\
&= \mathsf{Q}x\,.\alpha([\sigma, x/x](F')) \\
&= \mathsf{Q}x\,.[\![\sigma, x/x]\!](\alpha(F')) \\
&= [\![\sigma]\!](\mathsf{Q}x\,.(\alpha(F'))) \\
&= [\![\sigma]\!](\alpha(\mathsf{Q}x\,.F'))
\end{aligned}
$$

□

Unfortunately, the proof theory of second-order monadic logic is not defined in terms of flattening substitution application, but rather in terms of the standard form of application, as used in the quantifier rules in Fig. 2. However, there is a direct relationship between flattening substitution application and renaming substitutions ρ

$$\rho \quad ::= \quad \cdot \mid \rho, y/x \mid \rho, c/x.$$

A renaming ρ can only substitute variables or constants for variables because no successor functions are available.

This relationship is captured by extending the notion of abstraction α that currently maps only atomic formulas into conjunctions of monadic S0S predicates, to map substitutions σ into renaming substitutions ρ. Intuitively, $\alpha(\sigma)$ computes the witness substitution for the S0S quantifier rules.

$$
\begin{aligned}
\alpha(\cdot) &= \cdot \\
\alpha(\sigma, t/x) &= \alpha(\sigma), y/x \quad \text{for some } y \in VC(\sigma(x))
\end{aligned}
$$

If σ maps x to t, the corresponding ρ maps x to some variable or constant that occurs in t. Substitution abstraction is hence a necessary step to embed substitutions that arise in first-order logic derivations in S0S, but is it the right choice? Does it preserve the derivability of abstracted sequents?

The answer to this question is contingent on a suitable choice of abstraction to first-order logic derivations that we describe inductively. Abstracting a derivation tree proceeds by replacing each formula in the tree by its abstraction. Axioms $\Gamma, P(t_1, \ldots, t_n) \vdash P(t_1, \ldots, t_n), \Delta$, for example, are mapped into $\alpha(\Gamma), \alpha(P(t_1, \ldots, t_n)) \vdash \alpha(P(t_1, \ldots, t_n)), \alpha(\Delta)$. It remains to show that the abstracted derivation is really an S0S derivation which we do in two steps.

First, we show that the choice of renaming substitution is well chosen and compatible with the previous notion of flattening substitution application (see Definition 2.4). In the interest of brevity, we write $[\![\Gamma]\!]$ for a context that consists of $[\![\sigma_1]\!]F_1 \ldots [\![\sigma_n]\!]F_n$, and $[\Gamma]$ for a context of the form $[\alpha(\sigma_1)]F_1 \ldots [\alpha(\sigma_n)]F_n$. Second, we prove soundness of our abstraction.

Lemma 2.6 (Compatibility) *If $[\![\Gamma]\!] \Longrightarrow [\![\Delta]\!]$ is the result of abstracting a derivation then $[\Gamma] \Longrightarrow [\Delta]$.*

Proof. By induction on the derivation of $\Gamma \Longrightarrow \Delta$. The proof is quite straightforward. We only show three representative cases.

Case: $\dfrac{}{[\![\Gamma]\!], [\![\sigma]\!]P \Longrightarrow [\![\sigma]\!]P, [\![\Delta]\!]}\ \mathsf{ax}$

Similarly, we obtain $[\Gamma], [\sigma]P \Longrightarrow [\sigma]P, [\Delta]$ by the ax *rule.*

Case: $\dfrac{[\![\Gamma]\!], \forall x.\, [\![\sigma, x/x]\!]F, [t/x][\![\sigma, x/x]\!]F \Longrightarrow [\![\Delta]\!]}{[\![\Gamma]\!], \forall x.\, [\![\sigma, x/x]\!]F \Longrightarrow [\![\Delta]\!]}\ \forall\mathsf{L}.$

Since we are considering substitutions in S0S, the term t must always be a variable or a constant. By renaming we obtain that $[t/x][\![\sigma, x/x]\!]F = [\![\sigma, t/x]\!]F$, on which we can apply the induction hypothesis.

$$[\Gamma], \forall x.\, [\sigma, x/x]F, [\sigma, t/x]F \Longrightarrow [\Delta]$$

We can always rewrite the formula $[\sigma, t/x]F$ as $[t/x][\sigma, x/x]F$ by factoring out the renaming substitution and a renewed application of $\forall\mathsf{L}$ *yields the desired*

$$[\Gamma], \forall x.\, [\sigma, x/x]F \Longrightarrow [\Delta]$$

Case: $\dfrac{[\![\Gamma]\!] \Longrightarrow [a/x][\![\sigma, x/x]\!]F[\![\Delta]\!]}{[\![\Gamma]\!] \Longrightarrow, \forall x.\, [\![\sigma, x/x]\!]F[\![\Delta]\!]}\ \forall\mathsf{R}^a.$

As above, by renaming we obtain that $[a/x][\![\sigma, x/x]\!]F = [\![\sigma, a/x]\!]F$, on which we can apply the induction hypothesis.

$$[\Gamma] \Longrightarrow [\sigma, a/x]F, [\Delta]$$

We can always rewrite the term $[\sigma, a/x]F$ as $[a/x][\sigma, x/x]F$ by factoring out the renaming substitution. After discharging the parameter a, a renewed application of $\forall\mathsf{L}^a$ *yields the desired*

$$[\Gamma] \Longrightarrow \forall x.\, [\sigma, x/x]F, [\Delta]$$

□

The translation into monadic second-order logic reduces an intrinsically undecidable problem to a decidable one and allows us to conclude from the disproof of an abstracted conjecture that the original conjecture could not have been true. The following theorem establishes that relationship with the benefit that it defines implicitly a procedure to disprove false conjectures: Using the abstraction, convert a conjecture from first-order logic into second-order monadic logic, and then run an implementation of a decision procedure for S0S. This insight can be seen as the central contribution of this work.

Theorem 2.7 (Soundness) *The abstraction α of derivations in first-order logic into derivations of first-order monadic logic without successor functions preserves the non-provability of formulas: If $\Gamma \Longrightarrow \Delta$ then $\alpha(\Gamma) \Longrightarrow \alpha(\Delta)$.*

Proof. *By induction on the derivation of $\Gamma \Longrightarrow \Delta$. We only show the two challenging cases for the universal quantifier. All others are analogous.*

Case: $\dfrac{\Gamma \Longrightarrow \Delta, \forall x.\, F, [a/x]F}{\Gamma \Longrightarrow \Delta, \forall x.\, F}\ \forall\mathsf{I}^a$:

$\alpha(\Gamma) \Longrightarrow \alpha(\Delta, \forall x.\, F, [a/x]F)$	*by induction hypothesis*
$\alpha(\Gamma) \Longrightarrow \alpha(\Delta, \forall x.\, F), [\![a/x]\!]\alpha(F)$	*by Lemma 2.5*
$\alpha(\Gamma) \Longrightarrow \alpha(\Delta, \forall x.\, F), [a/x]\alpha(F)$	*by Lemma 2.6*
$\alpha(\Gamma) \Longrightarrow \alpha(\Delta, \forall x.\, F)$	*by* $\forall\mathsf{R}$

Case: $\dfrac{\Gamma, \forall x.\, F, [t/x]F \Longrightarrow \Delta}{\Gamma, \forall x.\, F \Longrightarrow \Delta}\ \forall\mathsf{E}$:

$\alpha(\Gamma, \forall x.\, F, [t/x]F) \Longrightarrow \alpha(\Delta)$	*by induction hypothesis*
$\alpha(\Gamma, \forall x.\, F), [\![t/x]\!]\alpha(F) \Longrightarrow \alpha(\Delta)$	*by Lemma 2.5*
$\alpha(\Gamma, \forall x.\, F), [\alpha(t/x)]\alpha(F) \Longrightarrow \alpha(\Delta)$	*by Lemma 2.6*
$\alpha(\Gamma, \forall x.\, F) \Longrightarrow \alpha(\Delta)$	*by* $\forall\mathsf{L}$

$\square$

Example 2.8 (Mirrored subtrees) Let F_0 be the conjunction of all axioms from Fig. 3 and $\alpha(F_0)$ the conjunction of all axioms from Fig. 5. Recall the problem from Example 1.4 of proving that a reflected path p in a mirrored tree t' leads to the same subtree as mirroring the subtree s that is found at p in the original tree t.

$$\begin{aligned}
F_0 \supset\ & \forall t.\, \forall s.\, \forall p.\, (\mathsf{validtree}(t) \wedge \mathsf{validtree}(s) \wedge \mathsf{validpath}(p) \wedge \mathsf{subtree}(t, p, s)) \\
& \supset \exists t'.\, \exists s'.\, \exists p'.\, (\mathsf{validtree}(t') \wedge \mathsf{validtree}(s') \wedge \mathsf{validpath}(p') \wedge \mathsf{subtree}(t', p', s') \\
& \wedge \mathsf{mirror}(t, t') \wedge \mathsf{reflect}(p, p') \wedge \mathsf{mirror}(s, s'))
\end{aligned}$$

In second-order monadic logic without successors the abstracted version of this formula is not provable either.

$$\begin{aligned}
F_0 \supset\ & \forall t.\, \forall s.\, \forall p.\, (\mathsf{validtree}(t) \wedge \mathsf{validtree}(s) \wedge \mathsf{validpath}(p) \\
& \quad \wedge \mathsf{subtree}(t) \wedge \mathsf{subtree}(p) \wedge \mathsf{subtree}(s)) \\
& \supset \exists t'.\, \exists s'.\, \exists p'.\, (\mathsf{validtree}(t') \wedge \mathsf{validtree}(s') \wedge \mathsf{validpath}(p') \\
& \wedge \mathsf{subtree}(t') \wedge \mathsf{subtree}(p') \wedge \mathsf{subtree}(s') \\
& \wedge \mathsf{mirror}(t) \wedge \mathsf{mirror}(t') \wedge \mathsf{reflect}(p) \wedge \mathsf{reflect}(p') \wedge \mathsf{mirror}(s) \wedge \mathsf{mirror}(s'))
\end{aligned}$$

Consequently, there is no need to invoke a first-order theorem prover, because by Theorem 2.7 it is determined to fail. On the other hand with induction, analyzing cases over p yields three conjectures whose abstractions are all provable in S0S assuming a few necessary but simple lemmas about binary trees and their abstractions, which we omit from this presentation. $\square$

The abstraction has many applications. For example, by trial and error it can be helpful to determine which axioms are indispensable for proof search. We also suspect that the proof derivations of the abstracted formula contains much information that is useful to guide a theorem prover during the proof search process.

3 Treating Primitive Equality

The decision procedure defined in the previous sections is restricted to first-order logic without primitive equality. Thus, equality is treated like any other binary predicate and an equation $s = t$ is abstracted to the monadic formula $(\bigwedge_{x \in VC(s=t)} = (x))$.

In order to support primitive equality in an adequate way we extend the abstraction function to primitive equality and abstract equations to

$$\begin{aligned}\alpha(s = t) := \ & \forall X. \left(\bigwedge_{x \in VC(s)} X(x)\right) \supset \left(\bigwedge_{x \in VC(t)} X(x)\right) \\ & \wedge \forall X. \left(\bigwedge_{x \in VC(t)} X(x)\right) \supset \left(\bigwedge_{x \in VC(s)} X(x)\right)\end{aligned}$$

Differently to the first-order case without equality, second-order quantifiers are necessary to range over predicates, such as $\mathsf{subtree}$, mirror, or $\mathsf{reflect}$.

Remark 3.1 This mapping is inspired by the Leibniz' definition of equality in higher-order logic, which is $s =_{Leibniz} t := \forall P. P(s) \supset P(t)$ with the only difference that besides the covariant it also involves the contravariant direction of implication. Without $\forall X. \left(\bigwedge_{x \in VC(t)} X(x)\right) \supset \left(\bigwedge_{x \in VC(s)} X(x)\right)$, for example, primitive equality would not be adequately captured in S0S. In higher-order logic P may be instantiated with any predicate $p_{\iota \to o}$ as well as with $\lambda x. \neg p(x)$, while in S0S the latter is not possible. However, the latter is necessary in order to obtain for each p not only $p(s) \supset p(t)$, but also the converse $p(t) \supset p(s)$, as used in the base case of Lemma 3.2.

It can be easily seen that the abstraction of a first-order equation is a second-order monadic formula due to the quantifier over X.

In the presence of primitive equality, we add the following rules to complete the sequent calculus for first-order logic with primitive equality. For those rules we denote by $C_{|u \leftarrow v}$ the replacement of exactly one occurrence of u with v in C.

$$\frac{}{\Gamma \Longrightarrow t = t, \Delta}\ \mathsf{refl} \qquad \frac{\Gamma, s = t \Longrightarrow F_{|t \leftarrow s}, \Delta}{\Gamma, s = t \Longrightarrow F, \Delta}\ \mathsf{Sub}^r_l \qquad \frac{\Gamma, s = t \Longrightarrow F_{|s \leftarrow t}, \Delta}{\Gamma, s = t \Longrightarrow F, \Delta}\ \mathsf{Sub}^r_r$$

$$\frac{\Gamma, F_{|t \leftarrow s}, s = t \Longrightarrow \Delta}{\Gamma, F, s = t \Longrightarrow \Delta}\ \mathsf{Sub}^l_l \qquad \frac{\Gamma, F_{|s \leftarrow t}, s = t \Longrightarrow \Delta}{\Gamma, F, s = t \Longrightarrow \Delta}\ \mathsf{Sub}^l_r$$

where for Sub-rules none of the variables in s and t are bound in F.

Lemma 3.2 *Any S0S sequent of the form $\Gamma, \alpha(s = t), \alpha(F_{|s \leftarrow t}) \Longrightarrow \alpha(F), \Delta$ or $\Gamma, \alpha(s = t), \alpha(F) \Longrightarrow \alpha(F_{|s \leftarrow t}), \Delta$ is provable.*

Proof. The proof is by induction over the structure of F.

Base Case: *$F = P(t_1, \ldots, t_n)$: In that case it holds*

$$\alpha(F_{|s \leftarrow t}) = \alpha(P(t_1, \ldots, t_n)_{|s \leftarrow t}) = \bigwedge_{x \in VC(P(t_1, \ldots, t_n)_{|s \leftarrow t})} P(x)$$

and $\alpha(F) = \alpha(P(t_1,\ldots,t_n)) = \bigwedge_{x\in VC(P(t_1,\ldots,t_n))} P(x)$. *Note that by definition of* VC *there exist lists* L, L' *such that* $VC(P(t_1,\ldots,t_n)_{|s\leftarrow t}) = L \oplus VC(t) \oplus L'$ *and* $VC(P(t_1,\ldots,t_n)) = L \oplus VC(s) \oplus L'$. *Furthermore,*

$$\begin{aligned}\alpha(s=t) = \ & \forall X.\ \left(\bigwedge\nolimits_{x\in VC(t)} X(x)\right) \supset \left(\bigwedge\nolimits_{x\in VC(s)} X(x)\right)\\ & \wedge\forall X.\ \left(\bigwedge\nolimits_{x\in VC(s)} X(x)\right) \supset \left(\bigwedge\nolimits_{x\in VC(t)} X(x)\right)\end{aligned}$$

By instantiating the first X *with* P *and the observation that* $VC(s)$ *is a sublist of* $VC(P(t_1,\ldots,t_n))$ *and* $VC(t)$ *is a sublist of* $VC(P(t_1,\ldots,t_n)_{|s\leftarrow t})$, *it is trivial to see that there is a proof for*

$$\begin{aligned}& \Gamma, \forall X.\ \left(\bigwedge\nolimits_{x\in VC(t)} X(x)\right) \supset \left(\bigwedge\nolimits_{x\in VC(s)} X(x)\right)\\ & \wedge\forall X.\ \left(\bigwedge\nolimits_{x\in VC(s)} X(x)\right) \supset \left(\bigwedge\nolimits_{x\in VC(t)} X(x)\right), \bigwedge\nolimits_{x\in VC(P(t_1,\ldots,t_n)_{|s\leftarrow t})} P(x)\\ \Longrightarrow\ & \bigwedge\nolimits_{x\in VC(P(t_1,\ldots,t_n))} P(x), \Delta\end{aligned}$$

The case for $\Gamma, \alpha(s=t), \alpha(F) \Longrightarrow \alpha(F_{|s\leftarrow t}), \Delta$ *is analogous, except that we must instantiate the second* X. *This is were the adequacy of the abstraction of an equation to both* $\forall X.\ \left(\bigwedge_{x\in VC(s)} X(x)\right) \supset \left(\bigwedge_{x\in VC(t)} X(x)\right)$ *and* $\forall X.\ \left(\bigwedge_{x\in VC(t)} X(x)\right) \supset \left(\bigwedge_{x\in VC(s)} X(x)\right)$ *is formally visible.*

Induction Step: *We proceed by case analysis over the structure of* F:

1. $F = \neg F'$: *It is obvious to see that* $\alpha(\neg(F')_{|s\leftarrow t}) = \neg(\alpha(F'_{|s\leftarrow t}))$. *Then*

$$\dfrac{\dfrac{\dfrac{}{\Gamma', \alpha(s=t), \alpha(F') \Longrightarrow \alpha(F'_{|s\leftarrow t}), \Delta}\ \text{I.H.}}{\Gamma', \neg(\alpha(F'_{|s\leftarrow t})), \alpha(s=t), \alpha(F') \Longrightarrow \Delta}\ \neg\mathsf{L}}{\Gamma', \neg(\alpha(F'_{|s\leftarrow t})), \alpha(s=t) \Longrightarrow \neg\alpha(F'), \Delta}\ \neg\mathsf{R}$$

2. $F = F_1 \wedge F_2$: *Without loss of generality we assume that* s *occurs in* F_1. *Again, it is obvious to see that* $\alpha((F_1 \wedge F_2)_{|s\leftarrow t}) = \alpha(F_{1|s\leftarrow t}) \wedge \alpha(F_2)$. *Then we have to prove* $\Gamma', \alpha(F_{1|s\leftarrow t}) \wedge \alpha(F_2), \alpha(s=t) \Longrightarrow \alpha(F_1) \wedge \alpha(F_2), \Delta$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\begin{array}{l}\Gamma', \alpha(F_{1|s\leftarrow t}), \alpha(s=t)\\ \Longrightarrow\ \alpha(F_1), \Delta\end{array}}\ \text{I. H.}}{\begin{array}{l}\Gamma', \alpha(F_{1|s\leftarrow t}), \alpha(F_2), \alpha(s=t)\\ \Longrightarrow\ \alpha(F_1), \Delta\end{array}}\ \text{weak L}}{\begin{array}{l}\Gamma', \alpha(F_{1|s\leftarrow t}) \wedge \alpha(F_2), \alpha(s=t)\\ \Longrightarrow\ \alpha(F_1), \Delta\end{array}}\ \wedge\mathsf{L} \quad \dfrac{}{\begin{array}{l}\Gamma', \alpha(F_{1|s\leftarrow t}), \alpha(F_2), \alpha(s=t)\\ \Longrightarrow\ \alpha(F_2), \Delta\end{array}}\ \mathsf{ax}}{\Gamma', \alpha(F_{1|s\leftarrow t}) \wedge \alpha(F_2), \alpha(s=t) \Longrightarrow \alpha(F_1) \wedge \alpha(F_2), \Delta}\ \wedge\mathsf{R}$$

3. $F = \forall x.\, F'$: *Again, it trivially holds that* $\alpha((\forall x.\, F')_{|s\leftarrow t}) = \forall x.\, \alpha(F'_{|s\leftarrow t})$. *Note that* x *does neither occur in* s *nor in* t. *Then we have to prove* $\Gamma', \forall x.\, \alpha(F'_{|s\leftarrow t}), \alpha(s = t) \Longrightarrow \forall x.\, \alpha(F'), \Delta$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma', \alpha([a/x]F'_{|s\leftarrow t}), \alpha(s=t) \Longrightarrow \alpha([a/x]F'), \Delta}\text{ I.H.}}{\Gamma', [\![a/x]\!]\alpha(F'_{|s\leftarrow t}), \alpha(s=t) \Longrightarrow [\![a/x]\!]\alpha(F'), \Delta}\text{ Lemma 2.5} \times 2}{\Gamma', [\alpha(a)/x]\alpha(F'_{|s\leftarrow t}), \alpha(s=t) \Longrightarrow [\alpha(a)/x]\alpha(F'), \Delta}\text{ Lemma 2.6} \times 2}{\Gamma', \forall x.\, \alpha(F'_{|s\leftarrow t}), [\alpha(a)/x]\alpha(F'_{|s\leftarrow t})\alpha(s=t) \Longrightarrow [\alpha(a)/x]\alpha(F'), \Delta}\text{ weak L}}{\Gamma', \forall x.\, \alpha(F'_{|s\leftarrow t}), \alpha(s=t) \Longrightarrow [\alpha(a)/x]\alpha(F'), \Delta}\,\forall\text{L}}{\Gamma', \forall x.\, \alpha(F'_{|s\leftarrow t}), \alpha(s=t) \Longrightarrow \forall x.\, \alpha(F'), \Delta}\,\forall\text{R}$$

4. *The remaining cases are analogous.* □

The soundness theorem with respect to first-order logic with primitive equality is then

Theorem 3.3 *The abstraction α of first-order logic formulas with primitive equality to second-order monadic logic formulas preserves the non-provability.*

Proof. Again we have to prove that whenever there is a proof for the original formula in first-order logic with primitive equality, then so there is for the abstracted formula. The proof is essentially the same as before, except that there are 5 additional cases to consider, one as an additional base case and four additional cases in the induction step:

Base Case: *Assume there is a derivation $\Gamma \vdash t = t, \Delta$, then we have to prove that there is a derivation for $\alpha(\Gamma) \vdash \alpha(t = t), \alpha(\Delta)$. Let $\varphi(X) := \bigwedge_{x \in VC(t)} X(x)$, then $\alpha(t = t) := (\forall X.\, \varphi(X) \supset \varphi(X)) \wedge \forall X.\, \varphi(X) \supset \varphi(X)$, and we must find a derivation in second-order monadic logic for $\alpha(\Gamma) \Longrightarrow (\forall X.\, \varphi(X) \supset \varphi(X)) \wedge (\forall X.\, \varphi(X) \supset \varphi(X)), \alpha(\Delta)$.*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha(\Gamma), \varphi(p) \vdash \varphi(p), \alpha(\Delta)}\text{ ax}}{\alpha(\Gamma) \Longrightarrow \varphi(p) \supset \varphi(p), \alpha(\Delta)} \supset\text{R}}{\alpha(\Gamma) \Longrightarrow \forall X.\, \varphi(X) \supset \varphi(X), \alpha(\Delta)}\,\forall\text{R}^p \quad \dfrac{\dfrac{\dfrac{}{\alpha(\Gamma), \varphi(p) \Longrightarrow \varphi(p), \alpha(\Delta)}\text{ ax}}{\alpha(\Gamma) \Longrightarrow \varphi(p) \supset \varphi(p), \alpha(\Delta)} \supset\text{R}}{\alpha(\Gamma) \Longrightarrow \forall X.\, \varphi(X) \supset \varphi(X), \alpha(\Delta)}\,\forall\text{R}^p}{\alpha(\Gamma) \Longrightarrow (\forall X.\, \varphi(X) \supset \varphi(X)) \wedge (\forall X.\, \varphi(X) \supset \varphi(X)), \alpha(\Delta)}\,\wedge\text{R}$$

Induction Step:

1. *Assume there is a derivation for $\Gamma, s = t \Longrightarrow F_{|s\leftarrow t}, \Delta$ and by induction hypothesis we can assume that there is an S0S derivation D for $\alpha(\Gamma), \alpha(s = t) \Longrightarrow \alpha(F_{|s\leftarrow t}), \alpha(\Delta)$. Then we have to prove that there is a second-order monadic logic derivation for $\alpha(\Gamma), \alpha(s = t) \Longrightarrow \alpha(F), \alpha(\Delta)$.*

$$\dfrac{\dfrac{}{\begin{array}{l}\alpha(\Gamma), \alpha(s=t), \alpha(F_{|s\leftarrow t}) \\ \Longrightarrow \alpha(F), \alpha(\Delta)\end{array}}\text{ Lemma 3.2} \quad \dfrac{\dfrac{}{\begin{array}{l}\alpha(\Gamma), \alpha(s=t) \\ \Longrightarrow \alpha(F_{|s\leftarrow t}), \alpha(\Delta)\end{array}}\text{ I.H.}}{\begin{array}{l}\alpha(\Gamma), \alpha(s=t) \\ \Longrightarrow \alpha(F_{|s\leftarrow t}), \alpha(F), \alpha(\Delta)\end{array}}\text{ weak R}}{\alpha(\Gamma), \alpha(s=t) \Longrightarrow \alpha(F), \alpha(\Delta)}\text{ Cut}(\alpha(F_{|s\leftarrow t}))$$

2. *The proofs for the other cases are analogous.* □

Example 3.4 Let F_0, and $\alpha(F_0)$ as in Example 2.8. A formula in first-order logic that concludes that any subtree in a tree t at path p is unique is

$$F_0 \supset \forall p.\, \forall p'.\, \forall t.\, \forall s.\, \forall s'.\, \mathsf{subtree}(t, p, s) \wedge \mathsf{subtree}(t, p', s') \supset s = s'.$$

Its abstraction expands the equality predicate as described above.

$$\begin{aligned} F_0 \supset\ & \forall p.\, \forall p'.\, \forall t.\, \forall s.\, \forall s'.\, \mathsf{subtree}(t) \wedge \mathsf{subtree}(p) \wedge \mathsf{subtree}(s) \\ & \wedge \mathsf{subtree}(t) \wedge \mathsf{subtree}(p') \wedge \mathsf{subtree}(s') \\ & \supset (\forall X.\, X(s) \supset X(s')) \wedge (\forall X.\, X(s') \supset X(s))\,. \end{aligned}$$

The resulting formula is not provable in S0S and can therefore not be proved in first-order logic with primitive equality by Theorem 3.3. On the other hand with induction, if one would consider cases over p, abstraction yields three cases, each of which is provable in S0S. □

4 About the Subclass of Unprovable Formulas

The question now arises which class of false conjectures can be tackled by the presented technique. Although we have no formal characterization for that class of formulas, we know that it includes first-order logic formulas that have only infinite counter-models. To see this consider the non-valid first-order logic formula in Fig. 6 and assume $\mathcal{I}$ is a counter-model that falsifies that formula. Then $\mathcal{I}(\varphi) = \bot$ entails that (1) $\mathcal{I}$ validates the left-hand side of the implication and (2) falsifies $\exists x. \neg P(x)$. From (1) it follows that the interpretations of P, $>$, and $=$ must be infinite. A possible infinite interpretation for P is $\lambda x. \top$. The abstraction

$$\begin{aligned} \text{FOL formula: } \varphi := \quad & (\exists x.\, P(x) \wedge \forall x.\, \exists y.\, P(x) \supset (y > x \wedge P(y)) \wedge \\ & \forall x, y, z.\, (x > y \wedge y > z) \supset x > z \wedge \forall x.\, x \neq x) \supset \exists x. \neg P(x) \\ \text{S0S formula: } \alpha(\varphi) := \quad & (\exists x.\, P(x) \wedge \forall x.\, \exists y.\, P(x) \supset (>(y) \wedge >(x) \wedge P(y)) \wedge \\ & \forall x, y, z.\, (>(x) \wedge >(y) \wedge >(z)) \supset >(x) \wedge >(z) \wedge \\ & \forall x.\, \neg(\forall X.\, X(x) \supset X(x) \wedge \forall X.\, X(x) \supset X(x))) \supset \exists x. \neg P(x) \end{aligned}$$

Figure 6: Disproven first-order logic formula with infinite counter-model.

$\alpha(\varphi)$ is also invalid with respect to S0S, also by interpreting P as $\lambda x. \top$. Thus, with our technique we can disprove first-order logic formulas that have no finite counter-models.

5 Implementation

The procedure for disproving false conjectures has been implemented in the MAYA system [3]. MAYA is an in-the-large verification tool for structured specifications. It is based on the notion of development graphs and incorporates an efficient management of change to preserve and adjust proof information when changing the specification. Each node of the development graph corresponds to an axiomatically defined theory and the procedure presented in this paper can be used to disprove false conjectures with respect to some theory. The implementation abstracts the first-order logic subset Φ of the axioms defining a theory to second-order monadic logic. To disprove a false conjecture ψ, the validity of the S0S formula $\alpha(\Phi \supset \psi)$ is checked.

In order to decide the validity of an S0S formula, rather than implementing our own S0S decision procedure, we have linked MAYA with the MONA system [7]. Although MONA implements only a decision procedure for *weak* second-order monadic logic, it is still useful since it is conservative over *full* second-order monadic logic without successor functions. Counter-models found in MONA are also counter-models in the more general setting. To our knowledge there is no available implementation of a full S0S decision procedure.

6 Conclusion

We have outlined a technique to disprove false conjectures in first-order logic with and without equality over a given and fixed term algebra. The central idea is that of abstraction. Formulas are transformed into second-order monadic logic without successor functions, which is known to be decidable. We have shown that the abstraction is sound, which means it preserves provability. Thus the absence of a proof in second-order monadic logic entails that the initial conjecture is unprovable, as well.

As related work we consider the tableau method [11] as well as combinations of model generation with automated theorem provers, such as the SCOTT system [6]. The tableau method not only detects unsatisfiability of the negated conjecture but also generates models for it. This is similar to the use of model generating systems during refutation proofs, as done in the SCOTT system. Thus, certain classes of false conjectures can be detected by generating counter-models. However, the relationship between these classes and the class characterized by the procedure presented in this paper is unclear yet and is left for future work.

Further future work is planned in different directions: First, we plan to investigate how to obtain from a counter-example for a non-valid S0S formula a counter-example for the original first-order logic formula, which would be highly beneficial especially in MAYA's application context which is formal software development. Also we assume it to be helpful to develop a characterization for the subclass of unprovable first-order logic formulas. Secondly, we plan to experiment with abstractions that preserve more of the term structures when mapping first-order logic formulas to second-order monadic logic formulas. Thereby we would leave the S0S fragment and employ larger fragments of second-order monadic logic, e.g. SkS. Preserving the structure should result in an increased efficiency for equational first-order logic theories. A third line of research will consist of using second-order logic proofs as proof plans to guide the actual proof search for the initial first-order logic formulas.

References

[1] P. B. Andrews, M. Bishop, and C. E. Brown. System Description: TPS: A Theorem Proving System for Type Theory. In D. McAllester, editor, *Proceedings of CADE-17*,

LNCS 1831, pages 164–169. Springer, 2000.

[2] S. Autexier, D. Hutter, B. Langenstein, H. Mantel, G. Rock, A. Schairer, W. Stephan, R. Vogt, and A. Wolpers. Vse: Formal methods meet industrial needs. *International Journal on Software Tools for Technology Transfer, Special issue on Mechanized Theorem Proving for Technology, Springer*, September 1998.

[3] S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The development graph manager MAYA. In H. Kirchner and C. Ringeissen, editors, *Proceedings 9th Int. Conference on Algebraic Methodology And Software Technology (AMAST'02)*, LNCS 2422. Springer, September 2002.

[4] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.

[5] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[6] K. Hodgson and J. Slaney. Development of a semantically guided theorem prover. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning*, LNAI 2083, pages 443–447. Springer, June 2001.

[7] N. Klarlund. Mona & fido: The logic-automaton connection in practice. In *Computer Science Logic, CSL '97*, LNCS 1414, 1998.

[8] G. Nadathur and D. Miller. An overview of λProlog. In K. A. Bowen and R. A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Washington, August 1988. MIT Press.

[9] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[10] C. Schürmann and S. Autexier. Towards proof planning for $\mathcal{M}_\omega^+$. *Electronic Notes in Theoretical Computer Science*, 70(2), 2002.

[11] R. Smullyan. *First-Order Logic*. Springer, 1968.

[12] J. Siekmann *et.al.* Proof development with ΩMEGA. In A. Voronkov, editor, *Proceedings of CADE-19*, LNAI 2392, pages 144–149, Copenhagen, Denmark, 2002. Springer.

[13] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the* 14^{th} *Annual ACM Symposium on Theory of Computing*, pages 137–146, 1982.