CHECKING VALIDITY OF QUANTIFIER-FREE FORMULAS IN COMBINATIONS OF FIRST-ORDER THEORIES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Clark Wayne Barrett
September 2002

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

David L. Dill
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Zohar Manna

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John Mitchell

Approved for the University Committee on Graduate Studies:

Abstract

An essential component in many verification methods is a fast decision procedure for validating logical expressions. This thesis presents several advances in the theory and implementation of such decision procedures, developed as part of ongoing efforts to improve the Stanford Validity Checker. We begin with the general problem of combining satisfiability procedures for individual theories into a satisfiability procedure for the combined theory. Two known approaches, those of Shostak and Nelson and Oppen, are described. We show how to combine these two methods to obtain the generality of the Nelson-Oppen method while retaining the efficiency of the Shostak method. We then present a general framework for combining decision procedures which includes features for enhancing performance and flexibility. Finally, validity checking requires that a heuristic search be built on top of the core decision procedure for satisfiability. We discuss strategies for efficient heuristic search and show how to adapt several powerful techniques from current research on Boolean satisfiability. Since these algorithms can be extremely subtle, a detailed proof of correctness is provided in the appendix.

Acknowledgments

I would like to thank my adviser David Dill for his support and guidance over many years. I would also like to thank the other members of my reading committee: Zohar Manna and John Mitchell. Many others have contributed to the success of this work, including Aaron Stump, Jeremy Levitt, Satyaki Das, Jeffrey Xsu, Robert Jones, Natarajan Shankar, Cesare Tinelli, SVC and CVC users, and my friends and family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An automated tool to check validity of formulas is of great interest because of its versatility. Many practical problems can be reduced to the question of whether some formula is valid in a given logical theory. Our experience with the Stanford Validity Checker (SVC) [1, 2], a tool for checking validity of quantifier-free formulas in a combination of first-order theories, confirms the need for and interest in such a tool. SVC has been used internally at Stanford for processor verification [23, 24, 25], symbolic simulation [39], software specification checking [32], and infinite-state model checking [11, 12]. In addition, since its public release in 1998, SVC has been downloaded and used in many other applications all over the world including model checking [6], theorem-prover proof assistance [21], programming language enhancements [13], and even the verification of an automobile airbag controller [4].

However, these applications revealed not only the need for such a tool, but also many limitations of the 1998 implementation. Our subsequent attempts to enhance and modify SVC revealed unnecessary constraints in the underlying theory, as well as gaps in our understanding of it. This thesis is an outcome of our attempt to re-architect SVC to resolve these difficulties. The primary goal has been to place SVC on a firm theoretical foundation without sacrificing the efficiency which made it successful.

Before describing the validity checking problem in more detail, we first give a brief overview of relevant concepts from first-order logic.

1.1 First-Order Logic

For those already familiar with basic first-order logic, this section may be skipped.

1.1.1 Basic Definitions

First-order logic is a widely used mathematical language for making precise statements [16, 22]. The "alphabet" of statements in first-order logic includes two distinct kinds of symbols: logical symbols and non-logical symbols. Logical symbols are common to all applications of first-order logic. They include parentheses, quantifiers, Boolean operators, and the equality operator. Note that although it is possible to use first-order logic without equality, we will make the assumption that equality is always included. Also, to simplify discussion, we will include the constant formulas *true* and *false* as logical symbols.

Non-logical symbols are symbols which vary depending on the application. They include variables, constant symbols, function symbols, and predicate (or relation) symbols. The symbols of first-order logic are summarized below.

1. Logical Symbols
   (a) Parentheses: (,)
   (b) Quantifiers: $\forall$ (for all), $\exists$ (there exists)
   (c) Boolean operators: $\neg$ (not), $\wedge$ (and), $\vee$ (or)
   (d) Constant formulas: *true*, *false*
   (e) Equality: $=$
2. Non-logical Symbols
   (a) Variables
   (b) Constant symbols
   (c) Function symbols: Each function symbol has an associated *arity*, a positive integer that indicates how many arguments it requires.

(d) Predicate symbols: Each predicate symbol also has an associated arity.

A symbol with arity 1, 2, or 3, is said to be a *unary*, *binary*, or *trinary* symbol respectively. In general, a symbol with arity n is said to be an n-ary symbol.

The "alphabet" of logical symbols and non-logical symbols is used to form *terms* and *formulas* (the rough logical equivalent of "words" and "phrases"). A *term* is a variable, a constant, or an application of an n-ary function symbol to n other terms. An *atomic formula* is either an equality between terms or an n-ary predicate symbol applied to n terms. A *literal* is either an atomic formula or its negation.

A *formula* is defined as follows.

(a) Constant formulas are formulas.

(b) Atomic formulas are formulas.

(c) If ϕ is a formula, then $\neg\phi$ is a formula.

(d) If ϕ and ψ are formulas, then $\phi \wedge \psi$ and $\phi \vee \psi$ are formulas.

(e) If ϕ is a formula and x is a variable, then $\forall x.\ \phi$ and $\exists x.\ \phi$ are formulas. In each case, x is said to be *bound* in ϕ.

Occurrences of variables which are not bound are said to be *free*. $\mathit{free}(\alpha)$ indicates the set of variables which occur free in α. If a formula ϕ contains no free variables (i.e. $\mathit{free}(\phi) = \emptyset$), it is called a *sentence*. If it contains no quantifiers, it is said to be *quantifier-free*.

Terms and formulas collectively are called *expressions*. To avoid confusion with the logical symbol $=$, we will use $\equiv$ to indicate that two logical expressions are identical. Similarly, when discussing Boolean values outside the context of logical expressions, `TRUE` and `FALSE` are used, so as not to confuse them with the logical formulas *true* and *false*.

Example 1.1. The non-logical symbols for an application involving simple arithmetic might be as follows.

1. Variables: $x, y, z, v_0, v_1, \ldots$

2. Constant symbols: $\ldots, -1, 0, 1, \ldots$

3. Function symbols: unary minus: $-$, binary plus: $+$

4. Predicate symbols: binary less-than: $<$, binary greater-than: $>$

Using these additional symbols, first-order logic can be used to precisely state various arithmetic properties:

1. Adding 0 to an integer does not change it: $\forall\, x.\, (x + 0 = x)$.

2. Addition is commutative: $\forall\, x.\, \forall\, y.\, (x + y = y + x)$.

3. There is always a larger number: $\forall\, x.\, \exists\, y.\, (y > x)$.

1.1.2 Theories and Models

A *theory* is a set of first-order sentences. For the purposes of this thesis, all theories are assumed to include the axioms of equality. The *signature* of a theory is the set of function, predicate, and constant symbols appearing in those sentences. The *language* of a signature Σ is the set of all expressions whose function, predicate, and constant symbols come from Σ. The language of a theory is the language of its signature. Terms or formulas in the language of a signature Σ are called Σ-*terms* or Σ-*formulas*. Given a signature Σ, a model M of Σ is a structure including the following four items:

1. A set called the *domain* of M, written $dom(M)$. Elements of the domain are called elements of the model M.

2. A mapping from each constant c in Σ to an element c^M of M.

3. A mapping from each n-ary function symbol f in Σ to f^M, an n-ary function from $(dom(M))^n$ to $dom(M)$.

4. A mapping from each n-ary predicate symbol p in Σ to p^M, an n-ary relation on the set $dom(M)$.

For a given model, M, a *variable assignment* ρ is a function which assigns to each variable an element of M. We say that M and ρ *satisfy* ϕ and write $M \models_\rho \phi$ if ϕ is true in the model M with variable assignment ρ. A formula ϕ is *satisfiable* if there exists some model M and variable assignment ρ such that $M \models_\rho \phi$. If Γ is a set of formulas and ϕ is a formula, then $\Gamma \models \phi$ means that for every model and variable assignment satisfying each formula in Γ, the same model and variable assignment also satisfy ϕ. A formula ϕ is *valid* if all models and variable assignments satisfy ϕ (i.e. $\emptyset \models \phi$). If $\mathcal{T}$ is a theory, we say ϕ is *valid in* $\mathcal{T}$ if $\mathcal{T} \models \phi$. Often, we will use sets of formulas where a logical formula is expected. The intended meaning is the conjunction of the formulas in the set. The conjunction of an empty set of formulas is defined to *true*.

A set S of literals is *convex* in a theory $\mathcal{T}$ if $\mathcal{T} \cup S$ does not entail any disjunction of equalities between variables without entailing one of the equalities itself. A theory $\mathcal{T}$ is convex if every set of literals in the language of the theory is convex in $\mathcal{T}$. A theory $\mathcal{T}$ is *stably infinite* if any quantifier-free formula is satisfiable in some model of $\mathcal{T}$ iff it is satisfiable in a model of $\mathcal{T}$ whose domain is infinite.

1.1.3 Validity Checking

Given a theory $\mathcal{T}$ and a formula ϕ, the validity checking problem is simply the problem of determining whether $\mathcal{T} \models \phi$. It is a well-known fact that in general, this problem is undecidable. There are at least two possible approaches to dealing with this undecidability. The first is to apply heuristics which will work well on some problems but give no result on others. The other is to restrict $\mathcal{T}$ and ϕ in such a way that the problem becomes decidable. Because we want a decision procedure, and because the applications in which we are interested can be handled without using the full expressive power of first-order logic, we take the second approach.

In the restricted domain we consider, ϕ is required to be quantifier-free, and the theory $\mathcal{T}$ is required to be the union of one or more theories $\mathcal{T}_i$ whose signatures are pairwise disjoint. Additionally, in each theory $\mathcal{T}_i$, the question of whether $\mathcal{T}_i \models \psi$, where ψ is a quantifier-free formula in the language of $\mathcal{T}_i$ must be decidable.

1.2 Some History

SVC has its roots in a ground-breaking paper by Burch and Dill on processor verification [9]. In order to better motivate the need for a validity checker, a brief overview of its role in the Burch-Dill verification methodology is given.

1.2.1 The Burch-Dill Method

The Burch-Dill method is used to verify that an *implementation* of a piece of hardware matches its *specification*. It also requires an *abstraction function* for matching implementation states with their corresponding specification states.

More concretely, suppose that for a given verification problem, Q_i is the set of possible states for the implementation and Q_s is the set of possible states for the specification, and that Abs is a function from Q_i to Q_s. Furthermore, suppose that F_i is the implementation transition function (from Q_i to Q_i) and that F_s is the specification transition function (From Q_s to Q_s).

In order to verify that the implementation is correct (with respect to the specification), consider a single transition starting from an arbitrary implementation state q_i. The result, $F_i(q_i)$, should correspond to a single transition of the specification starting from $Abs(q_i)$. In other words, the implementation is correct if

1.1. $F_s(Abs(q_i)) = Abs(F_i(q_i))$.

A diagram of this correctness condition is shown in Figure 1.1.

Traditional simulation techniques attempt to verify equation 1.1 for as many states in Q_i as possible. Unfortunately, for large and complex designs, exhaustive coverage is impossible. *Formal verification* techniques take a different approach: they attempt to *prove* that equation 1.1 is valid.

The theory in which this proof takes place depends on the transition and abstraction functions. Burch and Dill proposed a theory which included uninterpreted functions and predicates. They found that uninterpreted functions could be used to represent portions of the datapath that were common to both the implementation and the specification, greatly reducing the difficulty of checking formula 1.1. Thus,

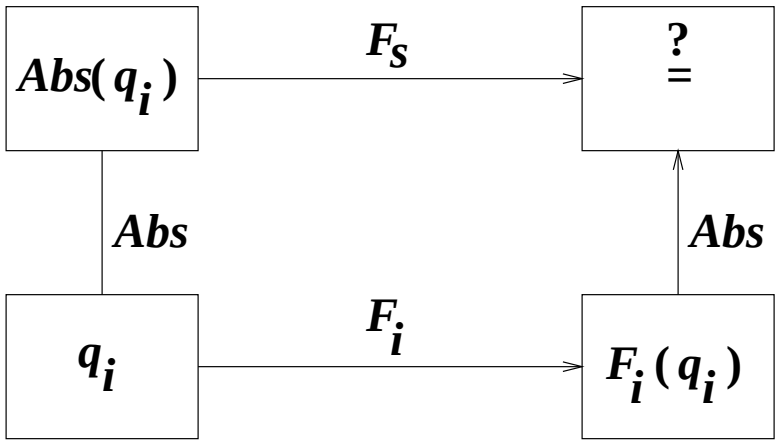

Figure 1.1: Burch-Dill Commuting Diagram

the first incarnation of what later became SVC was a simple validity checker for the logic of pure equality with uninterpreted functions.

1.2.2 The Evolution of SVC

This first validity checker was successfully applied to several designs [25]. Though these results were impressive, attempts to extend this initial work to more difficult designs revealed that the simple theory of pure equality with uninterpreted functions was insufficient. As time went on, a number of additional *interpreted* functions were added, including arithmetic, array, and bit-vector functions [1, 3].

The addition of these functions increased the expressive power of the logic. However, it also required a strategy for *combining* decision procedures for individual theories. The 1998 release of SVC included a number of theories combined in a somewhat *ad hoc* way, loosely based on Shostak's method for combining theories [1, 27, 35].

As mentioned, the 1998 release, though very successful by many standards, presented a number of difficulties. Perhaps the most fundamental difficulty was that our understanding of the underlying theory was limited. Decision procedures were required to conform to rigid requirements which limited the kinds of theories that could be included. Some theories which did not meet these requirements were included anyway, with the result that our confidence in the correctness of the entire

```
CheckValid(h,c)
  IF c ≡ true THEN RETURN TRUE;
  IF ¬Satisfiable(h) THEN RETURN TRUE;
  IF c ≡ false THEN RETURN FALSE;
  subgoals := ApplyTactic(h, c);
  FOREACH (h', c') ∈ subgoals DO
    IF ¬CheckValid(h', c') THEN RETURN FALSE;
  RETURN TRUE;
```

Figure 1.2: Top-level Validity Checking Algorithm

system was diminished.

There were other difficulties with the 1998 release. The system had outgrown its original software architecture with the result that attempts to modify or extend it in any way often broke the system in unpredictable ways. In particular, it was difficult to experiment with certain algorithmic changes which had the potential to dramatically increase performance.

This thesis is the result of efforts to address these issues. The theoretical and architectural contributions contained herein form the foundation for the successor to SVC, a system called CVC (Cooperating Validity Checker), which is significantly more robust, while remaining as useful as its predecessor.

1.3 Validity Checking: Top-Level Algorithm

In order to better understand the theoretical issues involved in validity checking, as well as to lay the foundation for a more detailed implementation, we here describe at a high level the algorithm used to check validity.

Figure 1.2 shows pseudocode for `CheckValid`, the top-level validity checking algorithm. The formula whose validity is to be determined is provided to `CheckValid` in the form of two arguments: a set h of formulas, the *hypotheses*, and a formula c, the *conclusion*. The pair (h, c) is called a *sequent*, and implicitly represents the formula $h \rightarrow c$. If the sequent is not obviously true or false, `CheckValid` applies a *tactic* which returns a set of subgoals (also represented as sequents). If the set is empty, that means the tactic has successfully proved the sequent. Otherwise, `CheckValid` is

```
ApplyTactic(h, c)
  Let φ be an atomic formula appearing in c;
  h1 := AddFact(h, φ);
  c1 := Simplify(h1, c);
  h2 := AddFact(h, ¬φ);
  c2 := Simplify(h2, c);
  RETURN {(h1, c1), (h2, c2)};
```

Figure 1.3: Case-Splitting Tactic

called recursively on each subgoal. A failure to prove any subgoal indicates that the original formula is not valid.

The strategy of representing goals as sequents and using tactics to break them down into subgoals is a common one used by many other theorem provers including HOL [20] and PVS [31]. Obviously, there is a lot of freedom in the choice of which tactics to use and how to choose between them. We focus on one particular tactic, a case-splitting tactic, which is sufficient to produce a decision procedure. Figure 1.3 shows pseudocode for a case-splitting tactic. This tactic makes use of two additional subroutines: `AddFact` and `Simplify`. `AddFact`(h, ϕ) simply adds ϕ to the hypotheses in h, and `Simplify`(h, c) simplifies c with respect to h[1].

This tactic is sufficient as long as `Simplify` satisfies certain conditions. A call to `Simplify`(h, c) must replace each atomic formula ϕ in c for which ϕ or $\neg\phi$ appears in h by *true* or *false* respectively. It is also required to evaluate purely propositional sentences to either *true* or *false*. Given these requirements, it is not hard to see that by repeatedly applying the case-splitting tactic, any quantifier-free formula will eventually reduce to true or false. Since the case-splitting tactic can only be applied a finite number of times to a finite formula, this means that the algorithm will always terminate.

Notice that our validity checking algorithm is built on top of an algorithm for determining the *satisfiability* of a conjunction of literals. This is the problem addressed in the bulk of the thesis. In particular, an efficient implementation of the functions

[1]From a proof-theoretic point of view, this simple top-level algorithm implements a classical Gentzen-style deductive system, and the case-splitting tactic is essentially a version of the *cut* inference rule.

`AddFact`, `Simplify`, and `Satisfiable` is the subject of chapter 3.

1.4 Organization

In chapter 2, we discuss the theoretical issues involved in combining satisfiability procedures for different theories. Two approaches which have been used in the past are those of Shostak [35] and Nelson-Oppen [29]. We present and prove the correctness of three new algorithms. The first is a simple subset of Shostak's algorithm. The next is a variation on Nelson-Oppen. The last is an instance of the Nelson-Oppen variation which generalizes the original Shostak algorithm.

Chapter 3 describes a detailed and efficient framework for combining theories. The framework includes a number of important features, such as a simplifier and a flexible interface for adding new theories. We show how the framework can be used to efficiently implement the abstract algorithms of chapter 2. We also describe a couple of important extensions which make the framework more powerful and give greater flexibility when adding new theories.

Chapter 4 revisits the top-level case-splitting algorithm and shows how it can be improved by using techniques from related work on Boolean satisfiability. We describe the results of combining our new system, CVC, with the Chaff Satisfiability solver [28]. The result is a combined system which on average requires far fewer decisions to successfully check the validity of formulas.

Chapter 5 summarizes the contributions of this thesis, offers some observations on verification, and describes future work, including one promising extension which can handle quantified formulas in some cases.

Finally, the appendix contains a detailed proof of the correctness of the algorithm presented in chapter 3.

Chapter 2

Combining Satisfiability Procedures

In this chapter, we consider the problem of determining the satisfiability of a conjunction of literals in a combined theory, given a satisfiability procedure for each theory individually. Two main approaches to this problem have emerged: Shostak's method and the Nelson-Oppen method. We will discuss each of these in turn.

In the process, we will also give a new simple presentation of Shostak's method without uninterpreted functions and a new variation of the Nelson-Oppen procedure. We then show how these two algorithms can be combined.

2.1 Shostak's Method

In 1984, Shostak introduced a clever and subtle algorithm which decides the satisfiability of quantifier-free formulas in a combined theory which includes a first-order theory (or combination of first-order theories) with certain properties and the theory of equality with uninterpreted function symbols [35]. But despite the fact that Shostak's method is less general than its predecessor, the Nelson-Oppen method [29, 30], it has generated considerable interest and is the basis for decision procedures found in several tools, including PVS [31], STeP [5, 17], and SVC [1, 2, 27].

There are several good reasons for this. First of all, it is easier to implement:

the Nelson-Oppen method provides a framework for combining decision procedures, but gives no help on how to construct the individual decision procedures. But as we will show below, at the core of Shostak's procedure is a simple recipe for generating decision procedures for a large class of theories. A second reason for the success of Shostak's method is that despite requiring more restrictive conditions in order to accommodate a theory, a wide variety of useful theories have been shown to satisfy these conditions [5, 35]. Finally, empirical studies have claimed that implementations based on Shostak's method are up to an order of magnitude more efficient than the Nelson-Oppen method [10].

Unfortunately, the original paper describing Shostak's method is difficult to follow, due in part to the fact that it contains several errors, and despite an ongoing effort to understand and clarify the method [10, 34, 42], it remains difficult to understand.

In the past, Shostak's algorithm has been presented either as a monolithic whole or as an extension of an algorithm for deciding just the the theory of pure equality with uninterpreted functions. We take a different approach by presenting a simple new algorithm (Algorithm *S1* below) based on a subset of Shostak's algorithm, in particular, the subset which decides formulas *without* uninterpreted functions. This algorithm provides considerable insight into how Shostak's algorithm works. It is also interesting in its own right because it is easily proved correct and can be used directly to produce decision procedures. Finally, the simplified algorithm forms the basis for a more general algorithm described in Section 2.3. A few definitions are required before proceeding.

2.1.1 Equations in Solved Form

A set $\mathcal{S}$ of equations is said to be in *solved form* iff the left-hand side of each equation in $\mathcal{S}$ is a variable which appears only once in $\mathcal{S}$. We will refer to these variables which appear only on the left-hand sides as *solitary* variables. A set $\mathcal{S}$ of equations in solved form defines an idempotent substitution: the one which replaces each solitary variable with its corresponding right-hand side. If S is an expression or set of expressions, we denote the result of applying this substitution to S by $\mathcal{S}(S)$. Another interesting

property of equations in solved form is that the question of whether such a set $\mathcal{S}$ entails some formula ϕ in a theory $\mathcal{T}$ can be answered simply by determining the validity of $\mathcal{S}(\phi)$ in $\mathcal{T}$.

Proposition 2.1. *If $\mathcal{T}$ is a theory with signature Σ and $\mathcal{S}$ is a set of Σ-equations in solved form, then $\mathcal{T} \cup \mathcal{S} \models \phi$ iff $\mathcal{T} \models \mathcal{S}(\phi)$.*

Proof. Clearly, $\mathcal{T} \cup \mathcal{S} \models \phi$ iff $\mathcal{T} \cup \mathcal{S} \models \mathcal{S}(\phi)$. Thus we only need to show that $\mathcal{T} \cup \mathcal{S} \models \mathcal{S}(\phi)$ iff $\mathcal{T} \models \mathcal{S}(\phi)$. The "if" direction is trivial. To show the other direction, assume that $\mathcal{T} \cup \mathcal{S} \models \mathcal{S}(\phi)$. Any model of $\mathcal{T}$ can be made to satisfy $\mathcal{T} \cup \mathcal{S}$ by assigning any value to the non-solitary variables of $\mathcal{S}$, and then choosing the value of each solitary variable to match the value of its corresponding right-hand side. Since none of the solitary variables occur anywhere else in $\mathcal{S}$, this assignment is well-defined and satisfies $\mathcal{S}$. By assumption then, this model and assignment also satisfy $\mathcal{S}(\phi)$, but none of the solitary variables appear in $\mathcal{S}(\phi)$, so the initial arbitrary assignment to non-solitary variables must be sufficient to satisfy $\mathcal{S}(\phi)$. Thus it must be the case that every model of $\mathcal{T}$ satisfies $\mathcal{S}(\phi)$ with every variable assignment. □

By setting ϕ to *false*, the following corollary is obtained.

Corollary 2.1. *If $\mathcal{T}$ is a satisfiable theory with signature Σ and $\mathcal{S}$ is a set of Σ-equations in solved form, then $\mathcal{T} \cup \mathcal{S}$ is satisfiable.*

2.1.2 Algorithm S1

We first give the conditions that a theory must meet in order for Algorithm *S1* to be applicable. We call a theory that meets these conditions a *Shostak* theory.

Definition 2.1. *A consistent theory $\mathcal{T}$ with signature Σ is a* Shostak *theory if the following conditions hold.*

1. *Σ does not contain any predicate symbols.*
2. *$\mathcal{T}$ is convex.*

3. *There exists a* canonizer *canon, a computable function from* Σ*-terms to* Σ*-terms, with the property that* $\mathcal{T} \models a = b$ *iff* $canon(a) \equiv canon(b)$.

4. *There exists a* solver *solve, a computable function from* Σ*-equations to sets of formulas defined as follows:*

   (a) *If* $\mathcal{T} \models a \neq b$, *then* $solve(a = b) \equiv \{false\}$.

   (b) *Otherwise,* $solve(a = b)$ *returns a set* $\mathcal{S}$ *of equations in solved form such that* $\mathcal{T} \models [(a = b) \leftrightarrow \exists \overline{w}.\, \mathcal{S}]$, *where* $\overline{w}$ *is the set of variables which appear in* $\mathcal{S}$ *but not in* a *or* b. *Each of these variables must be fresh.*

These requirements are slightly different from those given by Shostak and others. These differences are discussed in Section 2.4 below. In the rest of this section, $\mathcal{T}$ is assumed to be a Shostak theory with signature Σ, canonizer *canon*, and solver *solve*. As we will show, the solver can be used to convert an arbitrary set of equations into a set of equations in solved form. The canonizer is used to determine whether a specific equality is entailed by a set of equations in solved form, as shown by the following proposition.

Proposition 2.2. *If* $\mathcal{S}$ *is a set of* Σ*-equations in solved form, then* $\mathcal{T} \cup \mathcal{S} \models a = b$ *iff* $canon(\mathcal{S}(a)) \equiv canon(\mathcal{S}(b))$.

Proof. By Proposition 2.1, $\mathcal{T} \cup \mathcal{S} \models a = b$ iff $\mathcal{T} \models \mathcal{S}(a) = \mathcal{S}(b)$. But $\mathcal{T} \models \mathcal{S}(a) = \mathcal{S}(b)$ iff $canon(\mathcal{S}(a)) \equiv canon(\mathcal{S}(b))$ by the definition of *canon*. □

Algorithm *S1* (shown in Fig. 2.1) makes use of the properties of a Shostak theory to check the joint satisfiability of an arbitrary set of equalities, Γ, and an arbitrary set of disequalities, Δ, in a Shostak theory with canonizer *canon* and solver *solve*. Since the satisfiability of any quantifier-free formula can be determined by first converting it to disjunctive normal form, it suffices to have a satisfiability procedure for a conjunction of literals. Since Σ contains no predicate symbols, all Σ-literals are either equalities or disequalities. Thus, Algorithm *S1* is sufficient for deciding the satisfiability of quantifier-free Σ-formulas. Termination of the algorithm is trivial since each step

```
S1(Γ, Δ, canon, solve)
1.  S := ∅;
2.  WHILE Γ ≠ ∅ DO BEGIN
3.    Remove some equality a = b from Γ;
4.    a* := S(a); b* := S(b);
5.    S* := solve(a* = b*);
6.    IF S* = {false} THEN RETURN FALSE;
7.    S := S*(S) ∪ S*;
8.  END
9.  IF canon(S(a)) ≡ canon(S(b)) for some a ≠ b ∈ Δ THEN RETURN FALSE;
10. RETURN TRUE;
```

Figure 2.1: Algorithm S1: based on a simple subset of Shostak's algorithm

terminates and each time line 3 is executed the size of Γ is reduced. The following lemmas are needed before proving correctness.

Lemma 2.1. *If $\mathcal{T}'$ is a theory, Γ and Θ are sets of formulas, and $\mathcal{S}$ is a set of equations in solved form, then for any formula ϕ, $\mathcal{T}' \cup \Gamma \cup \Theta \cup \mathcal{S} \models \phi$ iff $\mathcal{T}' \cup \Gamma \cup \mathcal{S}(\Theta) \cup \mathcal{S} \models \phi$.*

Proof. Follows trivially from the fact that $\Theta \cup \mathcal{S}$ and $\mathcal{S}(\Theta) \cup \mathcal{S}$ are satisfied by exactly the same models and variable assignments. □

Lemma 2.2. *If Γ is any set of formulas, then for any formula ϕ, and Σ-terms a and b,*

$$\mathcal{T} \cup \Gamma \cup \{a = b\} \models \phi \text{ iff } \mathcal{T} \cup \Gamma \cup \mathit{solve}(a = b) \models \phi.$$

Proof.
$\Rightarrow$: Given that $\mathcal{T} \cup \Gamma \cup \{a = b\} \models \phi$, suppose that $M \models_\rho \mathcal{T} \cup \Gamma \cup \mathit{solve}(a = b)$. It is easy to see from the definition of *solve* that $M \models_\rho a = b$ and hence by the hypothesis, $M \models_\rho \phi$.
$\Leftarrow$: Given that $\mathcal{T} \cup \Gamma \cup \mathit{solve}(a = b) \models \phi$, suppose that $M \models_\rho \mathcal{T} \cup \Gamma \cup \{a = b\}$. Then, since $\mathcal{T} \models (a = b) \leftrightarrow \exists \overline{w}.\ \mathit{solve}(a = b)$, there exists a modified assignment ρ^* which assigns values to all the variables in $\overline{w}$ and satisfies $\mathit{solve}(a = b)$ but is otherwise equivalent to ρ. Then, by the hypothesis, $M \models_{\rho^*} \phi$. But the variables in $\overline{w}$ are fresh variables, so they do not appear in ϕ, meaning that changing their values cannot affect whether ϕ is true. Thus, $M \models_\rho \phi$. □

Lemma 2.3. *If Γ, $\{a = b\}$, and $\mathcal{S}$ are sets of Σ-formulas, with $\mathcal{S}$ in solved form, and if $\mathcal{S}^* = solve(\mathcal{S}(a = b))$ then if $\mathcal{S}^* \neq \{false\}$, then for every formula ϕ, $\mathcal{T} \cup \Gamma \cup \{a = b\} \cup \mathcal{S} \models \phi$ iff $\mathcal{T} \cup \Gamma \cup \mathcal{S}^* \cup \mathcal{S}^*(\mathcal{S}) \models \phi$.*

Proof.

$\mathcal{T} \cup \Gamma \cup \{a = b\} \cup \mathcal{S} \models \phi$	$\Leftrightarrow$	$\mathcal{T} \cup \Gamma \cup \{\mathcal{S}(a = b)\} \cup \mathcal{S} \models \phi$	Lemma 2.1
	$\Leftrightarrow$	$\mathcal{T} \cup \Gamma \cup \mathcal{S}^* \cup \mathcal{S} \models \phi$	Lemma 2.2
	$\Leftrightarrow$	$\mathcal{T} \cup \Gamma \cup \mathcal{S}^* \cup \mathcal{S}^*(\mathcal{S}) \models \phi$	Lemma 2.1

□

Lemma 2.4. *During the execution of Algorithm S1, $\mathcal{S}$ is always in solved form.*

Proof. Clearly, $\mathcal{S}$ is in solved form initially. Consider one iteration. By construction, a^* and b^* do not contain any of the solitary variables of $\mathcal{S}$, and thus by the definition of *solve*, $\mathcal{S}^*$ doesn't either. Furthermore, if $\mathcal{S}^* = \{false\}$ then the algorithm terminates at line 6. Thus, at line 7, $\mathcal{S}^*$ must be in solved form. Applying $\mathcal{S}^*$ to $\mathcal{S}$ guarantees that none of the solitary variables of $\mathcal{S}^*$ appear in $\mathcal{S}$, so the new value of $\mathcal{S}$ is also in solved form. □

Lemma 2.5. *Let Γ_n and $\mathcal{S}_n$ be the values of Γ and $\mathcal{S}$ after the while loop in Algorithm S1 has been executed n times. Then for each n, and any formula ϕ, the following invariant holds: $\mathcal{T} \cup \Gamma_0 \models \phi$ iff $\mathcal{T} \cup \Gamma_n \cup \mathcal{S}_n \models \phi$.*

Proof. The proof is by induction on n. For $n = 0$, the invariant holds trivially. Now suppose the invariant holds for some $k \geq 0$. Consider the next iteration.

$\mathcal{T} \cup \Gamma_0 \models \phi$	$\Leftrightarrow$	$\mathcal{T} \cup \Gamma_k \cup \mathcal{S}_k \models \phi$	Induction Hypothesis
	$\Leftrightarrow$	$\mathcal{T} \cup \Gamma_{k+1} \cup \{a = b\} \cup \mathcal{S}_k \models \phi$	Line 3
	$\Leftrightarrow$	$\mathcal{T} \cup \Gamma_{k+1} \cup \mathcal{S}^* \cup \mathcal{S}^*(\mathcal{S}_k) \models \phi$	Lemmas 2.3 and 2.4
	$\Leftrightarrow$	$\mathcal{T} \cup \Gamma_{k+1} \cup \mathcal{S}_{k+1} \models \phi$	Line 7

□

Now we can show the correctness of Algorithm *S1*.

Theorem 2.1. *Suppose $\mathcal{T}$ is a Shostak theory with signature Σ, canonizer canon, and solver solve. If Γ is a set of Σ-equalities and Δ is a set of Σ-disequalities, then $\mathcal{T} \cup \Gamma \cup \Delta$ is satisfiable iff* $\texttt{S1}(\Gamma, \Delta, canon, solve) = \texttt{TRUE}$.

Proof. Suppose $\texttt{S1}(\Gamma, \Delta, \mathit{canon}, \mathit{solve}) = \texttt{FALSE}$. If the algorithm terminates at line 9, then, $\mathit{canon}(\mathcal{S}(a)) \equiv \mathit{canon}(\mathcal{S}(b))$ for some $a \neq b \in \Delta$. It follows from Proposition 2.2 and Lemma 2.5 that $\mathcal{T} \cup \Gamma \models a = b$, so clearly $\mathcal{T} \cup \Gamma \cup \Delta$ is not satisfiable. The other possibility is that the algorithm terminates at line 6. Suppose the loop has been executed n times and that Γ_n and $\mathcal{S}_n$ are the values of Γ and $\mathcal{S}$ at the end of the last loop. It must be the case that $\mathcal{T} \models a^* \neq b^*$, so $\mathcal{T} \cup \{a^* = b^*\}$ is unsatisfiable. Clearly then, $\mathcal{T} \cup \{a^* = b^*\} \cup \mathcal{S}_n$ is unsatisfiable, so by Lemma 2.1, $\mathcal{T} \cup \{a = b\} \cup \mathcal{S}_n$ is unsatisfiable. But $\{a = b\}$ is a subset of Γ_n, so $\mathcal{T} \cup \Gamma_n \cup \mathcal{S}_n$ must be unsatisfiable, and thus by Lemma 2.5, $\mathcal{T} \cup \Gamma$ is unsatisfiable.

Suppose on the other hand that $\texttt{S1}(\Gamma, \Delta, \mathit{canon}, \mathit{solve}) = \texttt{TRUE}$. Then the algorithm terminates at line 10. By Lemma 2.4, $\mathcal{S}$ is in solved form. Let $\overline{\Delta}$ be the disjunction of equalities equivalent to $\neg(\Delta)$. Since the algorithm does not terminate at line 9, $\mathcal{T} \cup \mathcal{S}$ does not entail any equality in $\overline{\Delta}$. Because $\mathcal{T}$ is convex, it follows that $\mathcal{T} \cup \mathcal{S} \not\models \overline{\Delta}$. Now, since $\mathcal{T} \cup \mathcal{S}$ is satisfiable by Corollary 2.1, it follows that $\mathcal{T} \cup \mathcal{S} \cup \Delta$ is satisfiable. But by Lemma 2.5, $\mathcal{T} \cup \Gamma \models \phi$ iff $\mathcal{T} \cup \mathcal{S} \models \phi$, so in particular $\mathcal{T} \cup \mathcal{S} \models \Gamma$. Thus $\mathcal{T} \cup \mathcal{S} \cup \Delta \cup \Gamma$ is satisfiable, and hence $\mathcal{T} \cup \Gamma \cup \Delta$ is satisfiable. □

2.1.3 An Example

Perhaps the most obvious example of a Shostak theory is the theory of linear arithmetic with signature $\{0, S, +\}$ (where S is the *successor* function) and domain the real numbers. Terms in this theory can be more conveniently represented by using some standard abbreviations: base 10 numerals instead of repeated applications of successor (i.e. 3 instead of $S(S(S(0)))$), multiplication by a constant instead of repeated applications of $+$ (i.e. $3x$ instead of $x + x + x$). Division by a non-zero constant and the use of unary minus can also be included since equations involving these operations can always be converted into equivalent equations without them.

A simple canonizer for this theory can be obtained by imposing an order on all variables (lexicographic or otherwise), and combining like terms. For example, $\mathit{canon}(z + 3y - x - 5z) \equiv -x + 3y + (-4z)$. Similarly, a solver can be obtained simply by solving for one of the variables in an equation.

A well-known method for obtaining a solution to a system of equations in this theory is simply to use Gaussian elimination and back-substitution. Interestingly, by using the solver and canonizer just described, Algorithm *S1* actually implements Gaussian elimination with back-substitution.

Consider the following system of equations:

$$\begin{aligned} x + 3y - 2z &= 1 \\ x - y - 6z &= 1 \end{aligned}$$

This system can be represented by a matrix and transformed to reduced row echelon form as follows.

$$\left(\begin{array}{ccc|c} 1 & 3 & -2 & 1 \\ 1 & -1 & -6 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|c} 1 & 3 & -2 & 1 \\ 0 & -4 & -4 & 0 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|c} 1 & 0 & -5 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right)$$

Compare this with running Algorithm *S1* on the same set of equations. The following table shows the values of Γ, $\mathcal{S}$, $\mathcal{S}(a = b)$, and $\mathcal{S}^*$ on each iteration of Algorithm *S1* starting with $\Gamma = \{x + 3y - 2z = 1, x - y - 6z = 1\}$:

Γ	$\mathcal{S}$	$\mathcal{S}(a = b)$	$\mathcal{S}^*$
$x + 3y - 2z = 1$ $x - y - 6z = 1$	$\emptyset$	$x + 3y - 2z = 1$	$x = 1 - 3y + 2z$
$x - y - 6z = 1$	$x = 1 - 3y + 2z$	$1 - 3y + 2z - y - 6z = 1$	$y = -z$
$\emptyset$	$x = 1 + 5z$ $y = -z$		

The substitution for x in the second iteration corresponds to using x as a pivot variable to produce a zero in the second row of the matrix. Similarly, the last execution of line 7 transforms $x = 1 - 3y + 2z$ into $x = 1 + 5z$, corresponding to the transformation of the first row of the matrix due to back-substitution. Notice that the final solution obtained by Algorithm *S1* is the same as that obtained from the matrix in reduced row echelon form.

To make the example a little more interesting, suppose a third equation is added:

$2x + 8y - 2z = 3$. Transforming the matrix yields:

$$\left(\begin{array}{ccc|c} 1 & 3 & -2 & 1 \\ 1 & -1 & -6 & 1 \\ 2 & 8 & -2 & 3 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|c} 1 & 3 & -2 & 1 \\ 0 & -4 & -4 & 0 \\ 0 & 2 & 2 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|c} 1 & 3 & -2 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right)$$

At this point, the last row indicates that the system of equations is unsatisfiable. Suppose that the same new equation is processed by Algorithm *S1*. Note that rather than restarting the algorithm, the new equation can be placed in Γ and the algorithm can continue from where it left off. This illustrates a very nice property of the algorithm: it is incremental. If a new equation is added to Γ after some of the equations have already been processed, the algorithm can continue without any difficulty. The result is as follows:

Γ	$\mathcal{S}$	$\mathcal{S}(a = b)$	$\mathcal{S}^*$
$2x + 8y - 2z = 3$	$x = 1 + 5z$ $y = -z$	$2(1 + 5z) + 8(-z) - 2z = 3$	*false*

The solver detects an inconsistency when it tries to solve the equation obtained after applying the substitution from $\mathcal{S}$. The solver indicates this by returning $\{false\}$, which results in the algorithm returning `FALSE`.

Finally, suppose that instead of the equation $2x + 8y - 2z = 3$, the disequality $y + x \neq x - z$ is added. This is handled by line 9 of the algorithm:

$$\begin{array}{rcccl} canon(\mathcal{S}(y + x)) & \equiv & canon(-z + 1 + 5z) & \equiv & 1 + 4z \\ canon(\mathcal{S}(x - z)) & \equiv & canon(1 + 5z - z) & \equiv & 1 + 4z \end{array}$$

Since $y + x \neq x - z \in \Delta$ and $canon(\mathcal{S}(y + x)) \equiv canon(\mathcal{S}(x - z))$, the algorithm returns `FALSE`.

There is no matrix analog to the case which includes the disequality. Algorithm *S1* may, in fact, properly be viewed as a generalization of Gaussian elimination. Not only can it handle disequalities, but it can also introduce fresh variables or equations when solving. Also, the set of function symbols can be richer than those provided

by a vector space. The key requirement is simply that an appropriate canonizer and solver exist.

2.1.4 Combining Shostak Theories

In [35], Shostak claims that two Shostak theories can always be combined to form a new Shostak theory. A canonizer for the combined theory is obtained simply by composing the canonizers from each individual theory. A solver for the combined theory is ostensibly obtained by repeatedly applying the solver for each theory (treating terms in other theories as variables) until a true variable is on the left-hand side of each equation in the solved form. This does in fact work for many theories, providing a simple and efficient method for combining Shostak theories. However, as pointed out in [27] and [34], the construction of the solver as described is not always possible. We do not address this issue here, but mention it as a question which warrants further investigation.

2.2 The Nelson-Oppen Combination Method

Nelson and Oppen [29, 30] described a method for combining decision procedures for theories which are stably infinite and have disjoint signatures. In this section, we assume $\mathcal{T}_1$ and $\mathcal{T}_2$ are two such theories with signatures Σ_1 and Σ_2 respectively (the generalization to more than two theories is straightforward). Furthermore, we let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ and $\Sigma = \Sigma_1 \cup \Sigma_2$. The Nelson-Oppen procedure decides the satisfiability in $\mathcal{T}$ of a set Φ of Σ-literals.

2.2.1 Tinelli and Harandi's Approach

There have been many detailed presentations of the Nelson-Oppen method. Tinelli and Harandi's approach is particularly appealing because it is rigorous and conceptually simple [40]. Here we give a brief review of the method based on their approach. First, a few more definitions are required.

Members of Σ_i, for $i = 1,2$ are called i-symbols. In order to associate all terms with some theory, each variable is also arbitrarily associated with either $\mathcal{T}_1$ or $\mathcal{T}_2$. A variable is called an i-variable if it is associated with $\mathcal{T}_i$ (note that an i-variable is *not* an i-symbol, as it is not a member of Σ_i). A Σ-term t is an i-term if it is an i-variable, a constant i-symbol, or an application of a functional i-symbol. An i-predicate is an application of a predicate i-symbol. An atomic i-formula is an i-predicate or an equality whose left term is an i-term. An i-literal is an atomic i-formula or the negation of an atomic i-formula. An occurrence of a j-term t in either a term or a literal is i-alien if $i \neq j$ and all super-terms (if any) of that occurrence of t are i-terms. An i-term or i-literal is *pure* if the only non-logical symbols it contains are i-symbols and variables (i.e. only variables occur as i-alien sub-terms).

Given an equivalence relation $\sim$, let $dom_\sim$ be the domain of the relation. We define the following sets of formulas induced by $\sim$:

$$\begin{aligned} E_\sim &= \{x = y \mid x, y \in dom_\sim \text{ and } x \sim y\} \\ D_\sim &= \{x \neq y \mid x, y \in dom_\sim \text{ and } x \not\sim y\} \\ Ar_\sim &= E_\sim \cup D_\sim. \end{aligned}$$

Let Ar be a set of equalities and disequalities. If $Ar = Ar_\sim$ for some equivalence relation $\sim$ with domain Λ, we call Ar an *arrangement* of Λ.

The first step in determining the satisfiability of Φ is to transform Φ into an equisatisfiable formula $\Phi_1 \wedge \Phi_2$ where Φ_i consists only of pure i-literals as follows. Let ψ be some i-literal in Φ in which a non-variable j-term t occurs i-alien. Replace all occurrences of t in ψ with a fresh j-variable z and add the equation $z = t$ to Φ. Repeat until every literal in Φ is pure. The literals can then easily be partitioned into Φ_1 and Φ_2. It is easy to see that Φ is satisfiable if and only if $\Phi_1 \wedge \Phi_2$ is satisfiable.

Now, let Λ be the set of all variables which appear in both Φ_1 and Φ_2. A simple version of the Nelson-Oppen procedure simply guesses an equivalence relation $\sim$ on Λ nondeterministically, and then checks whether $\mathcal{T}_i \cup \Phi_i \cup Ar_\sim$ is satisfiable. The correctness of the procedure is based on the following theorem from [40].

Theorem 2.2. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably infinite, signature-disjoint theories and let Φ_i be a set of pure i-literals for $i = 1, 2$. Let Λ be the set of variables which appear*

in both Φ_1 *and* Φ_2. *Then* $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \Phi_1 \cup \Phi_2$ *is satisfiable iff there exists an arrangement* Ar *of* Λ *such that* $\mathcal{T}_i \cup \Phi_i \cup Ar$ *is satisfiable for* $i = 1, 2$.

2.2.2 A Variation of the Nelson-Oppen Procedure

The first step in the version of the Nelson-Oppen procedure described above changes the structure and number of literals in Φ. However, it is possible to give a version of the procedure which does not change the literals in Φ by instead treating occurrences of alien terms as variables. This simplifies the algorithm by eliminating the need for the purification step. But more importantly, this variation is required for the combination of Shostak and Nelson-Oppen described next.

First, we introduce a *purifying* operator which formalizes the notion of treating occurrences of alien terms as variables. Let v be a mapping from Σ-terms to variables such that for $i = 1, 2$, each i-term t is mapped to a *fresh* i-variable $v(t)$. Then, for some Σ-formula or Σ-term α, define $\gamma_i(\alpha)$ to be the result of replacing all i-alien occurrences of terms t by $v(t)$. It is easy to see that as a result, $\gamma_i(\alpha)$ is i-pure. Since γ_i simply replaces terms with unique place-holders, it is injective. We will denote its inverse by γ_i^{-1}. We will also denote by $\gamma_0(\alpha)$ the result of replacing each maximal term (i.e. terms without any super-terms) t in α by $v(t)$. Thus, the only terms in $\gamma_0(\alpha)$ are variables.

Our variation on the Nelson-Oppen procedure works as follows. Given a set of literals, Φ, first partition Φ into two sets Φ_1 and Φ_2, where Φ_i is exactly the set of i-literals in Φ. Let Λ be the set of all terms which occur i-alien (for some i) in some literal in Φ or in some sub-term of some literal in Φ. Λ consists of exactly those terms that would end up being replaced by variables in the original Nelson-Oppen method. Λ will also be referred to as the set of *shared* terms. As before, an equivalence relation $\sim$ on Λ is guessed. If $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_\sim)$ is satisfiable for each i, then $\mathcal{T} \cup \Phi$ is satisfiable, as shown by the following theorem.

Theorem 2.3. *Let* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *be two stably infinite, signature-disjoint theories and let* Φ *be a set of literals in the combined signature* Σ. *If* Φ_i *is the set of all* i*-literals in* Φ *and* Λ *is the set of shared terms in* Φ, *then* $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \Phi$ *is satisfiable iff there exists*

an equivalence relation $\sim$ *on* Λ *such that for* $i = 1, 2$, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_\sim)$ *is satisfiable.*

Proof.
$\Rightarrow$: Suppose $M \models_\rho \mathcal{T} \cup \Phi$. Let $a \sim b$ iff ($a, b \in \Lambda$ and $M \models_\rho a = b$). Then clearly for $i = 1, 2$, $M \models_\rho \mathcal{T}_i \cup \Phi_i \cup Ar_\sim$. It is then easy to see that $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_\sim)$ is satisfiable by choosing a variable assignment which assigns to each variable $v(t)$ the corresponding value of the term t which it replaces.
$\Leftarrow$: Suppose that for each i, $T_i \cup \gamma_i(\Phi_i \cup Ar_\sim)$ is satisfiable. Consider $i = 1$. Let Θ_1 be the set of all equations $v(t) = t$, where $t \in \Lambda$ is a 1-term. Consider $\gamma_1(\Theta_1)$. Since γ_1 never replaces 1-terms and each $v(t)$ is a fresh variable, it follows that $\gamma_1(\Theta_1)$ is in solved form, and its solitary variables are exactly the variables which are used to replace 1-terms. Thus, by Corollary 2.1, $\mathcal{T}_1 \cup \gamma_1(\Theta_1)$ is satisfiable. Furthermore, since none of the solitary variables of $\gamma_1(\Theta_1)$ appear in $\gamma_1(\Phi_1 \cup Ar_\sim)$, a satisfiable assignment for $\mathcal{T}_1 \cup \gamma_1(\Theta_1)$ can be constructed from the satisfying assignment for $\mathcal{T}_1 \cup \gamma_1(\Phi_1 \cup Ar_\sim)$ (which exists by hypothesis) so that the resulting assignment satisfies $\mathcal{T}_1 \cup \gamma_1(\Phi_1 \cup Ar_\sim \cup \Theta_1)$. Now, each term in $\gamma_1(Ar_\sim)$ which is not already a variable is the right-hand side of some equation in $\gamma_1(\Theta_1)$, so by repeatedly applying equations from $\gamma_1(\Theta_1)$ as substitutions, $\gamma_1(Ar_\sim)$ can be transformed into $\gamma_0(Ar_\sim)$, and thus $\mathcal{T}_1 \cup \gamma_1(\Phi_1 \cup \Theta_1) \cup \gamma_0(Ar_\sim)$ must also be satisfiable. Applying the same argument with i=2, we conclude that $\mathcal{T}_2 \cup \gamma_2(\Phi_2 \cup \Theta_2) \cup \gamma_0(Ar_\sim)$ is satisfiable. But for each i, $\gamma_i(\Phi_i \cup \Theta_i)$ is a set of i-literals. Furthermore, $\gamma_0(Ar_\sim)$ is an arrangement of the variables shared by these two sets, so Theorem 2.2 can be applied to conclude that $\mathcal{T} \cup \Phi \cup \Theta_1 \cup \Theta_2$, and thus $\mathcal{T} \cup \Phi$, is satisfiable. □

2.2.3 A Deterministic Implementation for Convex Theories

A deterministic version of our Nelson-Oppen variation for convex theories is shown in Fig. 2.2. Algorithm *N-O* takes as input a set of literals Φ (partitioned into 1-literals Φ_1 and 2-literals Φ_2) and two decision procedures, Sat_1 and Sat_2, where Sat_i decides the satisfiability of literals in $\mathcal{T}_i$. Formally, for any set of Σ_i-literals Θ,

```
N-O(Φ1, Sat1, Φ2, Sat2)
1.  done := FALSE;
2.  ∼ := reflexive-only relation on the shared terms in Φ1 ∪ Φ2;
3.  WHILE ¬done DO BEGIN
4.    done := TRUE;
5.    FOR i := 1,2 DO BEGIN
6.      IF ¬Sati(Φi ∪ E∼) THEN RETURN FALSE;
7.      IF ¬Sati(Φi ∪ Ar∼) THEN BEGIN
8.        Choose a ≠ b ∈ D∼ such that ¬Sati(Φi ∪ E∼ ∪ {a ≠ b});
9 .       ∼ := symmetric-transitive closure of ∼ ∪ (a, b).
10.       done := FALSE;
11.     END
12.   END
13. END
14. RETURN TRUE;
```

Figure 2.2: Algorithm N-O: an implementation of the Nelson-Oppen variation for two convex theories

$$Sat_i(\Theta) = \texttt{TRUE} \textit{ iff } \mathcal{T}_i \cup \gamma_i(\Theta) \not\models \textit{false}.$$

The algorithm seeks to discover an arrangement $Ar_\sim$ by successive refinement of $\sim$. Initially, $\sim$ is the reflexive-only relation on all shared terms. In each iteration, the satisfiability of $\Phi_i \cup Ar_\sim$ is checked for each theory $\mathcal{T}_i$. If both are satisfiable, then Φ is satisfiable and algorithm terminates. If the equalities in $Ar_\sim$ are sufficient to cause the unsatisfiability, then the algorithm terminates at line 6.

If not, then it is possible to choose a single disequality of $Ar_\sim$ which, in the presence of the equalities $E_\sim$ and the literals Φ_i, is unsatisfiable in $\mathcal{T}_i$. This is because if the algorithm reaches line 8, then $\gamma_i(\Phi_i \cup E_\sim \cup D_\sim)$ is not satisfiable in $\mathcal{T}_i$, but $\gamma_i(\Phi_i \cup E_\sim)$ is. It follows from convexity of $\mathcal{T}_i$ that there must be a disequality $a \neq b$ in $D_\sim$ such that $\gamma_i(\Phi_i \cup E_\sim \cup \{a \neq b\})$ is not satisfiable in $\mathcal{T}_i$. One simple implementation of this step is as follows. Start with a set of literals consisting of $\Phi_i \cup E_\sim$. Then, incrementally add disequalities from $D_\sim$ until the set becomes unsatisfiable. The last disequality added has the desired property.

It is easy to see that the algorithm terminates because each step terminates and

the loop can only be executed a finite number of times. To see why, notice that each time `done` is set to `FALSE`, two equivalence classes of $\sim$ are merged together. Since the domain of $\sim$ is finite (the shared terms of Φ) and does not increase, equivalence classes can only be merged a finite number of times.

Lemma 2.6. *Let $\sim_n$ be the value of $\sim$ after line 9 in Algorithm N-O has been executed n times. Then for each n, if $\approx$ is an equivalence on the shared terms of Φ such that for $i = 1, 2$, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_\approx)$ is satisfiable, then $E_{\sim_n} \subseteq Ar_\approx$.*

Proof. The proof is by induction on n. For $n = 0$, $E_{\sim_n}$ is empty, so the invariant holds trivially. Now assume it holds for k and consider the next execution of line 9. Suppose that $\approx$ is an equivalence relation on the shared terms such that for $i = 1, 2$, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_\approx)$ is satisfiable. By the induction hypothesis, $E_{\sim_k} \subseteq Ar_\approx$. By line 8, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_k} \cup \{a \neq b\})$ is not satisfiable. Since either $a = b$ or $a \neq b$ must be in $Ar_\approx$, it must be the case that $a = b \in Ar_\approx$. It follows that $E_{\sim_{k+1}} \subseteq Ar_\approx$. □

Theorem 2.4. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two convex, stably infinite, signature-disjoint theories and let Φ be a set of literals in the combined signature Σ. Furthermore, for $i = 1, 2$, let Sat_i be a procedure for deciding satisfiability of conjunctions of literals in $\mathcal{T}_i$ as defined above. If Φ_i is the set of all i-literals in Φ and Λ is the set of shared terms in Φ, then $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \Phi$ is satisfiable iff $\texttt{N-O}(\Phi_1, Sat_1, \Phi_2, Sat_2) = \texttt{TRUE}$.*

Proof. Suppose $\texttt{N-O}(\Phi_1, Sat_1, \Phi_2, Sat_2) = \texttt{FALSE}$. This can only happen if the algorithm terminates at line 6. Suppose there is an equivalence relation $\approx$ on the shared terms such that for $i = 1, 2$, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_\approx)$ is satisfiable. It follows from Lemma 2.6 that $E_\sim \subseteq Ar_\approx$. But since the algorithm terminates at line 6, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_\sim)$ must be unsatisfiable. Thus no such equivalence relation $\approx$ can exist. Thus, by Theorem 2.3, $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \Phi$ is unsatisfiable.

Suppose on the other hand that $\texttt{N-O}(\Phi_1, Sat_1, \Phi_2, Sat_2) = \texttt{TRUE}$. Then, it must be the case that the if condition in line 7 is false for both i=1 and i=2 the last time the while loop is executed. This means that $T_i \cup \gamma_i(\Phi_i \cup Ar_\sim)$ is satisfiable for $i = 1, 2$. Thus, by Theorem 2.3, $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \Phi$ is satisfiable. □

2.3 Combining Nelson-Oppen and Shostak

In order to embed Shostak's algorithm in the more general Nelson-Oppen framework, we use the following result which relates convexity (a requirement for Shostak theories) and stable infiniteness, (a requirement for applying the Nelson-Oppen method). Note that a trivial model is one whose domain contains only a single element.

Theorem 2.5. *A convex first-order theory with no trivial models is stably infinite.*

Proof. Suppose $\mathcal{U}$ is a first-order theory which is not stably infinite. Then there exists some quantifier-free set of literals Φ which is satisfiable in a finite model of $\mathcal{U}$, but not in an infinite model of $\mathcal{U}$. Let $\exists \overline{x}.\Phi$ be the existential closure of Φ. Then $\exists \overline{x}.\Phi$ is true in some finite model, but not in any infinite model, of $\mathcal{U}$. It follows that $\mathcal{U} \cup \{\exists \overline{x}.\ \Phi\}$ is a theory with no infinite models. By first-order compactness, there must be some finite cardinality n such that there is a model of $\mathcal{U} \cup \{\exists \overline{x}.\ \Phi\}$ of cardinality n, but none of cardinality larger than n. Clearly, $\mathcal{U} \cup \Phi$ is satisfiable in some model of size n, but not in any models larger than n. It follows by the pigeonhole principle that if $y_i, 0 \leq i \leq n$ are fresh variables, then $\mathcal{U} \cup \Phi \models \bigvee_{i \neq j} y_i = y_j$, but because $\mathcal{U}$ has no trivial models, $\mathcal{U} \cup \Phi \not\models y_i = y_j$ for any i, j with $i \neq j$. Thus, $\mathcal{U}$ is not convex. □

Now let $\mathcal{T}_1, \mathcal{T}_2, \Sigma_1, \Sigma_2, \mathcal{T}$, and Σ be defined as in the previous section, with the additional assumptions that $\mathcal{T}_1$ is a Shostak theory and that neither $\mathcal{T}_1$ nor $\mathcal{T}_2$ admits trivial models (typically, theories of interest do not admit trivial models, or can be easily modified so that this is the case). The above theorem implies that both theories are also stably infinite. As a result, they can be combined using the Nelson-Oppen method.

One obvious way to combine the two theories is simply to use Algorithm *N-O* with Algorithm *S1* as the Sat_1 parameter. Although this works, we next describe an algorithm which combines the two methods explicitly. Our purpose in doing this is to describe an algorithm which is still abstract enough that it can be understood and proved correct, but specific enough that it is not hard to see how to specialize it further to recover Shostak's original algorithm (this is described in Section 2.4.2, below). The combined algorithm not only sheds light on how Shostak's method can be seen as an

efficient refinement of the Nelson-Oppen method, but also provides a starting point for achieving other efficient refinements. Indeed, the next chapter describes a detailed implementation-level framework based on the combined algorithm.

2.3.1 The Combined Algorithm

Suppose Φ is a set of Σ-literals. As in Section 2.2.2, divide Φ into Φ_1 and Φ_2 where Φ_i contains exactly the i-literals of Φ. Let Λ be the set of shared terms. By Theorem 2.3, $\mathcal{T} \cup \Phi_1 \cup \Phi_2$ is satisfiable iff there exists an equivalence relation $\sim$ such that for $i = 1, 2$, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_\sim)$ is satisfiable.

In order for the approach in Algorithm *S1* to function in a multiple-theory environment, it is necessary to generalize the definition of equations in solved form to accommodate the notion of treating occurrences of alien terms as variables. A set $\mathcal{S}$ of equations is said to be in *i-solved form* if $\gamma_i(\mathcal{S})$ is in solved form. If $\mathcal{S}$ is a set of equations in i-solved form and Λ is an expression or set of expressions in a mixed language including Σ_i, then we define $\mathcal{S}(\Lambda)$ to be the result of replacing each i-alien occurrence in Λ of the left-hand sides of equations in $\mathcal{S}$ with the corresponding right-hand side. Formally, $\mathcal{S}(\Lambda)$ is redefined to be $\gamma_i^{-1}(\gamma_i(\mathcal{S})(\gamma_i(\Lambda)))$, i.e. the application of $\mathcal{S}$ to Λ should be equivalent to first replacing all i-alien occurrences of terms with variables in both $\mathcal{S}$ and Λ, then doing the substitution, and then finally restoring the i-alien terms to their places. We similarly need to extend the definitions of *canon* and *solve*. Let $canon_1(\alpha)$ denote $\gamma_1^{-1}(canon(\gamma_1(\alpha)))$ and $solve_1(\beta)$ denote $\gamma_1^{-1}(solve(\gamma_1(\beta)))$.

Now, let Γ be the set of all equalities in Φ_1 and Δ the set of disequalities in Φ_1. Furthermore, as in Algorithm *N-O*, above, let Sat_2 be a decision procedure for satisfiability of literals in $\mathcal{T}_2$:

$$Sat_2(\Phi) = \texttt{TRUE} \textit{ iff } \mathcal{T}_2 \cup \gamma_2(\Phi) \not\models false.$$

Algorithm *S2* combines Algorithms *S1* and *N-O* . Essentially, lines 3 through 5 mimic the Nelson-Oppen approach for $\mathcal{T}_2$, while the rest of the algorithm is identical to *S1*. Rather than being maintained explicitly as in Algorithm *N-O*, the equivalence relation $\sim$ on Λ is derived from $\mathcal{S}$:

```
S2(Γ, Δ, canon, solve, Φ2, Sat2)
1.  S := ∅;
2.  WHILE Γ = ∅ OR ¬Sat2(Φ2 ∪ Ar∼) DO BEGIN
3.    IF ¬Sat2(Φ2 ∪ Ar∼) THEN BEGIN
4.      IF ¬Sat2(Φ2 ∪ E∼) THEN RETURN FALSE;
5.      ELSE Choose a ≠ b ∈ D∼ such that ¬Sat2(Φ2 ∪ E∼ ∪ {a ≠ b});
6.    END ELSE Remove some equality a = b from Γ;
7.    a* := S(a); b* := S(b);
8.    S* := solve1(a* = b*);
9.    IF S* = {false} THEN RETURN FALSE;
10.   S := S*(S) ∪ S*;
11. END
12. IF a∼b for some a ≠ b ∈ Δ THEN RETURN FALSE;
13. RETURN TRUE;
```

Figure 2.3: Algorithm S2: a generalization of Shostak's algorithm

$$a \sim b \text{ iff } a, b \in \Lambda \wedge canon_1(\mathcal{S}(a)) \equiv canon_1(\mathcal{S}(b))$$

In each iteration of the while loop, an equation is processed and integrated with $\mathcal{S}$. This equation is either the result of the current arrangement being inconsistent in $\mathcal{T}_2$ (lines 3 through 5) or simply an equation from Γ (line 6). As shown below, the definition of $\sim$ ensures that $\mathcal{S}$ is consistent with $Ar_\sim$. Similarly, equations are added to $\mathcal{S}$ until $Ar_\sim$ is also consistent with Φ_2. Thus, when the algorithm returns `TRUE`, both Φ_1 and Φ_2 are known to be consistent with the arrangement $Ar_\sim$.

Algorithm *S2* terminates because each step terminates and in each iteration either the size of Γ is reduced by one or two equivalence classes in $\sim$ are merged. As before, the correctness proof requires a couple of preparatory lemmas.

Lemma 2.7. *Suppose $\mathcal{S}$ is a set of Σ-formulas in 1-solved form, Λ is a set of Σ-terms, and $\sim$ is defined as above. If $\approx$ is an equivalence relation on Λ such that $\mathcal{T}_1 \cup \gamma_1(Ar_\approx \cup \mathcal{S})$ is satisfiable, then $E_\sim \subseteq Ar_\approx$. In other words, every arrangement of Λ consistent with $\mathcal{S}$ must include $E_\sim$.*

Proof. Consider an arbitrary equation $a = b$ between terms in Λ. $a = b \in E_\sim$ iff $canon_1(\mathcal{S}(a)) \equiv canon_1(\mathcal{S}(b))$ iff (by Proposition 2.2) $\mathcal{T}_1 \cup \gamma_1(\mathcal{S}) \models \gamma_1(a = b)$. So

$\gamma_1(a = b)$ must be true in every model and assignment satisfying $\mathcal{T}_1 \cup \gamma_1(\mathcal{S})$. In particular, if $\mathcal{T}_1 \cup \gamma_1(Ar_\approx \cup \mathcal{S})$ is satisfiable, the corresponding model and assignment must also satisfy $\gamma_1(a = b)$. Since either the equation $a = b$ or the disequation $a \neq b$ must be in $Ar_\approx$, it must be the case that $a = b \in Ar_\approx$. Thus, $E_\sim \subseteq Ar_\approx$. □

Lemma 2.8. *Let Γ_n and $\mathcal{S}_n$ be the values of Γ and $\mathcal{S}$ after the loop in Algorithm S2 has been executed n times. Then for each n, the following invariant holds: $\mathcal{T} \cup \Phi$ is satisfiable iff there exists an equivalence relation $\approx$ on Λ such that*

(1) $\mathcal{T}_1 \cup \gamma_1(\Gamma_n \cup \Delta \cup Ar_\approx \cup \mathcal{S}_n)$ is satisfiable, and

(2) $\mathcal{T}_2 \cup \gamma_2(\Phi_2 \cup Ar_\approx)$ is satisfiable.

Proof. The proof is by induction on n. For the base case, notice that by Theorem 2.3, $\mathcal{T} \cup \Phi$ is satisfiable iff there exists an equivalence relation $\approx$ such that (1) and (2) hold with $n = 0$.

Before doing the induction case, we first show that for some fixed equivalence relation $\approx$, (1) and (2) hold when $n = k$ iff (1) and (2) hold when $n = k + 1$. Notice that (2) is independent of n, so it is only necessary to consider (1). There are two cases to consider.

First, suppose that the condition of line 3 is true and line 5 is executed. We first show that (1) holds when $n = k$ iff the following holds:

(3) $\mathcal{T}_1 \cup \gamma_1(\Gamma_{k+1} \cup \Delta \cup Ar_\approx \cup \{a = b\} \cup \mathcal{S}_k)$ is satisfiable.

Since line 6 is not executed, $\Gamma_{k+1} = \Gamma_k$. The if direction is then trivial since the formula in (1) is a subset of the formula in (3). To show the only if direction, first note that it follows from line 5 that $\mathcal{T}_2 \cup \gamma_2(\Phi_2 \cup E_\sim) \models \gamma_2(a = b)$. But by Lemma 2.7, $E_\sim \subseteq Ar_\approx$, so it follows that $\mathcal{T}_2 \cup \gamma_2(\Phi_2 \cup Ar_\approx) \models \gamma_2(a = b)$. Since either $a = b \in Ar_\approx$ or $a \neq b \in Ar_\approx$, it must be the case that $a = b \in Ar_\approx$ and thus (3) follows trivially from (1). Now, by Lemma 2.3 (where ϕ is false), if line 10 is reached, then (3) holds iff

(4) $\mathcal{T}_1 \cup \gamma_1(\Gamma_{k+1} \cup \Delta \cup Ar_\approx \cup \mathcal{S}^*(\mathcal{S}_k) \cup \mathcal{S}^*)$ is satisfiable,

where $\mathcal{S}^* = solve_1(\mathcal{S}(a = b))$. But $\mathcal{S}_{k+1} = \mathcal{S}^*(\mathcal{S}_k) \cup \mathcal{S}^*$, so (4) is equivalent to (1) with $n = k + 1$.

In the other case, line 6 is executed (so that $\Gamma_{k+1} = \Gamma_k - \{a = b\}$). Thus, (1) holds with $n = k$ iff $\mathcal{T}_1 \cup \gamma_1(\Gamma_{k+1} \cup \Delta \cup \{a = b\} \cup Ar_{\approx} \cup \mathcal{S}_k)$ is satisfiable, which is equivalent to (3). As in the previous case, it then follows from Lemma 2.3 that (1) holds at k iff (1) holds at $k + 1$.

Thus, given an equivalence relation, (1) and (2) hold at $k + 1$ exactly when they hold at k. It follows easily that if an equivalence relation exists which satisfies (1) and (2) at k, then there exists an equivalence relation satisfying (1) and (2) at $k + 1$ and vice-versa. Finally, the induction case assumes that $\mathcal{T} \cup \Phi$ is satisfiable iff there exists an equivalence relation $\approx$ such that (1) and (2) hold at k. It follows from the above argument that $\mathcal{T} \cup \Phi$ is satisfiable iff there exists an equivalence relation $\approx$ such that (1) and (2) hold at $k + 1$. □

Theorem 2.6. *Suppose that $\mathcal{T}_1$ is a Shostak theory with signature Σ_1, canonizer canon, and solver solve, and that $\mathcal{T}_2$ is a convex theory with signature Σ_2 disjoint from Σ_1 and satisfiability procedure Sat_2. Suppose also that neither $\mathcal{T}_1$ nor $\mathcal{T}_2$ admit trivial models, and let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ and $\Sigma = \Sigma_1 \cup \Sigma_2$. Suppose Φ is a set of Σ-literals. Let Γ be the subset of Φ which consists of 1-equalities, Δ the subset of Φ which consists of 1-disequalities, and Φ_2 the remainder of the literals in Φ. $\mathcal{T} \cup \Phi$ is satisfiable iff* $\texttt{S2}(\Gamma, \Delta, canon, solve, \Phi_2, Sat_2) = \texttt{TRUE}$.

Proof. First note that by the same argument used in Lemma 2.4, $\mathcal{S}$ is always in 1-solved form.

Suppose $\texttt{S2}(\Gamma, \Delta, canon, solve, \Phi_2, Sat_2) = \texttt{FALSE}$. If the algorithm terminates at line 9 or 12, then the proof that Φ is unsatisfiable is the same as that for Algorithm *S1* above. If it stops at line 4, then suppose there is an equivalence relation $\approx$ satisfying condition (1) of Lemma 2.8. It follows from Lemma 2.7 that $E_{\sim} \subseteq Ar_{\approx}$. But since the algorithm terminates at line 4, $\mathcal{T}_2 \cup \gamma_2(\Phi_2 \cup Ar_{\approx})$ must be unsatisfiable. Thus condition (2) of Lemma 2.8 cannot hold. Thus, by Lemma 2.8, $\mathcal{T} \cup \Phi$ is unsatisfiable.

Suppose on the other hand that $\texttt{S2}(\Gamma, \Delta, canon, solve, \Phi_2, Sat_2) = \texttt{TRUE}$. By the definition of $\sim$ and Proposition 2.2, $a = b \in Ar_{\sim}$ iff $\mathcal{T}_1 \cup \gamma_1(\mathcal{S}) \models \gamma_1(a = b)$. It follows

from the convexity of $\mathcal{T}_1$ and Corollary 2.1 that $\mathcal{T}_1 \cup \gamma_1(\mathcal{S} \cup Ar_\sim)$ is satisfiable. It then follows from the fact that *S2* does not terminate at line 12 (as well as convexity again) that $\mathcal{T}_1 \cup \gamma_1(\mathcal{S} \cup \Delta \cup Ar_\sim)$ is satisfiable. This is condition (1) of Lemma 2.8. Condition (2) must hold because the while loop terminates. Thus, by Lemma 2.8, $\mathcal{T} \cup \Phi$ is satisfiable. □

2.3.2 An Example

Let $\mathcal{T}_1$ be the theory of linear arithmetic described in Section 2.1.3 above. Let $\mathcal{T}_2$ be the pure theory of equality with uninterpreted functions. Consider the following formula from Shostak's original paper [35]:

$$z = f(x - y) \wedge x = z + y \wedge -y \neq -(x - f(f(z))).$$

For this example, if all variables are assumed to be 1-variables, we have the following:

$$\begin{array}{lcl} \Gamma & = & \{z = f(x - y), x = z + y\} \\ \Delta & = & \{-y \neq -(x - f(f(z)))\} \\ \Phi_2 & = & \emptyset \\ \Lambda & = & \{x - y, f(x - y), z, f(f(z))\} \end{array}$$

Recall that a term is shared if it occurs i-alien in either a literal or a sub-term of a literal in Φ. We will step through the execution of Algorithm *S2* on this example. The table below shows the values of Γ, $\mathcal{S}$, and the equivalence classes of $\sim$ for each iteration. On the first iteration, the test on line 3 fails, so line 6 is executed and $z = f(x-y)$ is chosen, which is already in 1-solved form, so $\mathcal{S}$ becomes $\{z = f(x-y)\}$. As a result, the equivalence classes containing z and $f(x - y)$ are merged, since $canon_1(\mathcal{S}(z)) \equiv canon_1(\mathcal{S}(f(x - y))) \equiv f(x - y)$. On the next iteration, $x = z + y$ is chosen. After applying $\mathcal{S}$ to the equation, we get $x = f(x - y) + y$, which is already solved, so it is added to $\mathcal{S}$. Now notice that $canon_1(\mathcal{S}(x-y)) \equiv canon(f(x - y) + y - y) \equiv f(x - y)$, so $x - y$ must be in the same equivalence class as $f(x - y)$ and z. At this point, $Ar_\sim$ includes $z = x - y, x - y = f(x - y)$, and $z \neq f(f(z))$.

These three formulas are not satisfiable in $\mathcal{T}_2$ since the first two imply the negation of the third. Thus $z \neq f(f(z))$ is chosen in line 5 and the algorithm continues. After executing line 8, $\mathcal{S}^* = \{f(x-y) = f(f(z))\}$, so as a result of executing line 9, $f(x-y)$ is replaced everywhere in $\mathcal{S}$ by $f(f(z))$. The loop exits since the while condition is no longer true. The final row of the table shows the final value of $\mathcal{S}$. Now, observe that $canon_1(\mathcal{S}(-(x - f(f(z))))) \equiv canon_1(-(f(f(z)) + y - f(f(z)))) \equiv -y \equiv canon_1(\mathcal{S}(-y))$. Thus, since $-y \neq -(x - f(f(z))) \in \Delta$, the algorithm will halt at line 12 and report that the formula is unsatisfiable.

Γ	$\mathcal{S}$	$\sim$
$z = f(x-y)$ $x = z + y$	$\emptyset$	$\{\{x-y\}, \{f(x-y)\}, \{z\}, \{f(f(z))\}\}$
$x = z + y$	$z = f(x-y)$	$\{\{x-y\}, \{f(x-y), z\}, \{f(f(z))\}\}$
$\emptyset$	$z = f(x-y)$ $x = f(x-y) + y$	$\{\{x-y, f(x-y), z\}, \{f(f(z))\}\}$
$\emptyset$	$z = f(f(z))$ $x = f(f(z)) + y$ $f(x-y) = f(f(z))$	$\{\{x-y, f(x-y), z, f(f(z))\}\}$

2.4 Comparison with Shostak's Original Method

There are two main ways in which this work differs from Shostak's original method, which is best represented by Ruess and Shankar in [34]. The first is in the set of requirements a theory must fulfill. The second is in the level of abstraction at which the algorithm is presented.

2.4.1 Requirements on the Theory

Of the four requirements given in our definition of a Shostak theory, the first two are clarifications which are either assumed or not addressed in other work, and the last two are similar to, but slightly less restrictive, than the requirements listed by others. The first requirement is simply that the theory contain no predicate symbols. This is

a minor point which is included simply to be explicit about an assumption which is implicit in other work. Shostak's method does not give any guidance on what to do if a theory includes predicate symbols. One possible approach is to encode predicates as functions, but this only works if the resulting encoding admits a canonizer and solver.

The second requirement is that the theory be convex. This may seem overly restrictive since Shostak claims that non-convex theories can be handled [35]. Consider, however, the following simple non-convex theory with signature $\{a, b\}$: $\{a \neq b, \forall\, x.\, (x = a \vee x = b)\}$. It is easy to see that this theory admits a (trivial) canonizer and a solver. However, for the unsatisfiable set of formulas $\{x \neq y, y \neq z, x \neq z\}$, any version of Shostak's algorithm will fail to detect the inconsistency. Ruess and Shankar avoid this difficulty by restricting their attention to the problem of whether $\mathcal{T} \cup \Gamma \models a = b$ for some set of equalities Γ. However, the ability to solve this problem does not lead to a self-contained decision procedure unless the theory is convex.

The third requirement on the theory is that a canonizer exist. Shostak gave several additional properties that must be satisfied by the canonizer. These are not needed at the level of abstraction of our algorithms, though some efficient implementations may require the additional properties.

A similar situation arises with the requirements on the solver: only a subset of the original requirements are needed. Note that although we require the set of equalities returned by the solver to be equisatisfiable with the input set in *every* model of $\mathcal{T}$, whereas Ruess and Shankar require only that it be equisatisfiable with the input set in every σ-model[1], it is not difficult to show that their requirements on the canonizer imply that every model of $\mathcal{T}$ must be a σ-model.

2.4.2 Level of Abstraction

Though Algorithm *S2* looks very different from Shostak's original published algorithm as well as most other published versions, it is, in fact, closely related to them, differing primarily in that it is more abstract. For example, an algorithm equivalent to that

[1] In the notation of Ruess and Shankar, the canonizer is denoted by σ, and a *σ-model* M is one where $M \models a = \sigma(a)$ for any term a.

found in [34] can be obtained by making a number of refinements. We do not describe these in detail, but we outline them briefly below. We also describe some general principles they exemplify which could be used in other refinements.

The most obvious refinement is to replace $\mathcal{T}_2$ by the theory of equality with uninterpreted function symbols. The data structure for $\mathcal{S}$ can be expanded to include all equations (not just the 1-equations), obviating the need to track Φ_2 separately. The check for satisfiability in $\mathcal{T}_2$ is replaced by a simple check for congruence closure over the terms in $\mathcal{S}$. The general principle here is that if $\mathcal{S}$ can be expanded to track the equalities in another theory, then equality information only needs to be maintained in one place, which is more efficient.

Another refinement is that a more sophisticated substitution can be applied at line 7 of Algorithm *S2*. The more sophisticated substitution considers each sub-term t, and if it is known to be equivalent to a term u already appearing in $\mathcal{S}$, then all instances of t are replaced with u. For terms in the Shostak theory, this is essentially accomplished by applying the canonizer. For uninterpreted function terms, it is a bit more subtle. For example, if $x = y \in \mathcal{S}$ and $f(x)$ appears in $\mathcal{S}$, then if $f(y)$ is encountered, it can be replaced by $f(x)$. As a result, fewer total terms are generated and thus fewer terms need to be considered when updating $\mathcal{S}$ or when performing congruence closure. The general principle is that simplifications and substitutions which reduce the total number of terms can improve efficiency. This is especially important in a natural generalization of Algorithm *S2* to accommodate non-convex theories in which the search for an appropriate arrangement of the shared terms can take time which is more than exponential in the number of shared terms [30].

Chapter 3

A Framework for Combining Theories

In this chapter, a concrete implementation framework is presented which allows satisfiability procedures for disjoint theories to cooperate. The implementation is based on algorithm *N-O* of the previous chapter but includes a number of additional features.

First of all, it is an *online* algorithm, meaning that instead of checking satisfiability of a set of formulas, the formulas are fed into the framework one at a time. The framework tracks whether the set of formulas seen so far is satisfiable, and only a small amount of incremental work is needed to process each new formula. An online algorithm is much more versatile and is especially useful in the context of our top-level validity checking algorithm which incrementally builds a set of formulas, checking their satisfiability after each additional formula. Note that with an online algorithm, the set of shared terms may grow over time. The framework is designed to handle this automatically.

Another feature of the framework is its use of a union-find data structure to efficiently maintain an equivalence relation on shared terms. The framework is structured in such a way as to allow this same data structure to represent also the set of equations in solved form which must be maintained for a Shostak theory. Using a single data structure keeps equality reasoning localized in one place avoiding redundant work.

Additionally, a simplification phase has been added to the algorithm. Conceptually, the simplifier applies quick and easy rewrite rules which can reduce the number of shared terms seen by the core algorithm. The simplifier can also enforce certain syntactical requirements on the terms which appear in the framework. For example, if a theory has a canonizer (as Shostak theories do), it can be applied during the simplification phase, ensuring that only the canonical form of each term appears. This makes it unnecessary to deduce and propagate equalities between terms with the same canonical form.

Finally, a flexible interface is provided for decision procedures for individual theories. This interface allows an individual theory to use the code and data structures provided by the framework (rather than having to provide its own) whenever possible. At the end of this chapter, we give several examples of how this interface can be used.

3.1 An Overview of the Framework

Suppose that $\mathcal{T}_1, \ldots, \mathcal{T}_N$ are N first-order theories, with signatures $\Sigma_1, \ldots, \Sigma_N$. Let $\mathcal{T} = \bigcup \mathcal{T}_i$ and $\Sigma = \bigcup \Sigma_i$. We assume that the intersection of any two signatures is empty and that each theory is stably infinite. The goal is to provide a framework for a satisfiability procedure which determines the satisfiability in $\mathcal{T}$ of a set of formulas in the language of Σ. This is done by maintaining an implicit set of formulas Φ (initially empty) which we call the *fact database* and reporting if that database ever becomes inconsistent.

As shown in Figure 3.1, the framework is intended for use within a context which includes three parts: user code, framework code, and theory-specific code. The user code is the code which calls the framework. It could be a simple user-interface to the framework, or it could be an algorithm built on top of the framework (such as the top-level algorithm of CVC described in chapter 1).

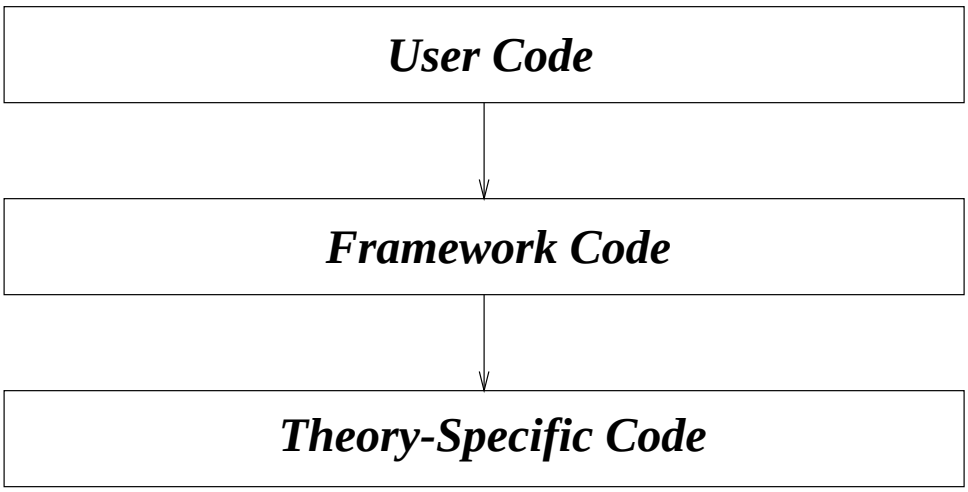

Figure 3.1: The Framework Context

3.1.1 The Interface to the User Code

The primary interface between the user code and the framework code is the procedure `AddFact`. The user calls `AddFact` in order to add a new formula to the fact database Φ. Another piece of the framework which is exported to the user is the `Simplify` procedure. If `Simplify` is called with argument e, the result is an expression which is equivalent to (and hopefully simpler than) e modulo $\mathcal{T} \cup \Phi$. Finally, at any time, the user can also call `Satisfiable` which returns `TRUE` iff the current fact database is consistent.

At this point, we should point out that the interface just described is slightly different from the one required by the top-level algorithm described in Section 1.3. The only difference is that the top-level algorithm of figures 1.2 and 1.3 passes an additional argument to each of the procedures `AddFact`, `Simplify`, and `Satisfiable`. This additional argument corresponds to the fact database which the framework maintains implicitly. In order to make the interface compatible, it is necessary for the framework to provide two additional procedures: `Save` and `Restore`. `Save` returns the current state of the framework, and `Restore` restores the state of the framework from a previously saved state. Figure 3.2 shows the modified top level algorithm corresponding to the interface described in this chapter.

```
CheckValid(h,c)
  IF c ≡ true THEN RETURN TRUE;
  Restore(h);
  IF ¬Satisfiable() THEN RETURN TRUE;
  IF c ≡ false THEN RETURN FALSE;
  subgoals := ApplyTactic(h, c);
  FOREACH (h, c) ∈ subgoals DO
    IF ¬CheckValid(h, c) THEN RETURN FALSE;
  RETURN TRUE;

ApplyTactic(h, c)
  Restore(h);
  Let φ be an atomic formula appearing in c;
  AddFact(φ);
  c1 := Simplify(c);
  h1 := Save();
  Restore(h);
  AddFact(¬φ);
  c2 := Simplify(c);
  h2 := Save();
  RETURN {(h1, c1), (h2, c2)};
```

Figure 3.2: Modified Top-level Validity Checking Algorithm

In a practical implementation, saving and restoring the entire fact database is impractical. Fortunately, the same functionality can be implemented using a stack as discussed by Jones [25]. Though this can be challenging to implement correctly, the basic concept is straightforward, so we will not address the implementation of `Save` and `Restore` here.

3.1.2 The Interface to the Theory-Specific Code

The interface between the framework code and the theory-specific code consists of several procedures which are parameterized by theory, meaning that there is an instance of each of them for each theory. We indicate which theory's instance should be called by subscripting the call with a theory index. Thus, if `f` is a theory-specific procedure, $\mathtt{f}_i$ denotes the instance of `f` associated with theory $\mathcal{T}_i$.

There are seven theory-specific procedures which can be called by the framework.

They are: `TheoryAddSharedTerm`, `TheoryAssert`, `TheoryCheckSat`, `TheoryRewrite`, `TheorySetup`, `TheorySolve`, and `TheoryUpdate`. These will be described in detail below.

3.1.3 A Comparison With Algorithm *N-O*

As mentioned, the framework is loosely based on Algorithm *N-O*. To see the connection, compare Algorithm *N-O* to the main procedure of the framework, `AddFact`, shown below.

```
AddFact(e)
  Q := {e};
  REPEAT
    WHILE Q ≠ ∅ AND ¬I DO BEGIN
      Choose e* ∈ Q;
      Q := Q − {e*};
      Assert(e*);
    END
    FOR i := 1 TO N DO
      IF Q = ∅ AND ¬I THEN TheoryCheckSat_i();
  UNTIL Q = ∅ OR I;
```

One main difference is that `AddFact` is online. Formulas are passed to `AddFact` one at a time. `Assert` is then responsible for assigning each new formula to one of the theories. `Assert` is also responsible for maintaining an equivalence relation $\sim$ on the terms shared among the sets Φ_i. The equivalent of lines 6 through 11 of algorithm *N-O* are handled by the theory-specific procedure $\texttt{TheoryCheckSat}_i$, which is required to check the satisfiability of Φ_i in the arrangement of the shared terms induced by $\sim$.

`AddFact` is described in more detail in Section 3.3 below. But before describing the framework in detail, we describe the data structures used by the framework.

3.2 Data structures

The most basic data structure in the framework is that used for representing logical *expressions*.

3.2.1 Expressions

As defined earlier, an *expression* is either a term or a formula. A (quantifier-free) expression is a *leaf* if it is a variable or constant. Otherwise, it can be viewed as a *compound* expression: an *operator* (a function, predicate, or Boolean connective) applied to one or more *children* (the operands of the operator). For a compound expression e, $Op(e)$ refers to the operator of e, $Arity(e)$ to the arity of the operator, and $e[i]$ to the i^{th} child of e, where $e[1]$ is the first child, and the children are numbered from left to right.

For example, suppose $e \equiv f(x)$, then $Op(e) \equiv f$ and $e[1] \equiv x$. On the other hand, if $e \equiv (t = f(f(z)))$, then $Op(e) \equiv$ '=', while $e[1] \equiv t$ and $e[2] \equiv f(f(z))$.

We assume that expressions are represented using a labeled directed acyclic graph (DAG) data structure as follows. If an expression e is a compound expression, then its DAG node is labeled with $Op(e)$, and for each child of e, there is an edge from the node for e to a node representing the child. We assume maximal subexpression sharing so that any two expressions which are syntactically identical are uniquely represented by a single DAG node. We will refer to expressions and their associated DAG nodes interchangeably.

We will continue to use the concepts and notation introduced in Section 2.2.1. Recall that an expression which is either a term or literal in the language of Σ is an i-expression if it is a variable associated with $\mathcal{T}_i$, its operator is a symbol in Σ_i, or it is an equality or disequality and its left-hand side is an i-term. We use the notation $\mathcal{T}(e)$ to refer to the theory associated with an expression. Thus, if e is an i-expression, $\mathcal{T}(e) \equiv i$. A j-term t occurs as an i-leaf in an expression e if every super-term of that occurrence (not including t) is an i-term and t is a variable or $i \neq j$. Note that a term t occurs as an i-alien of an expression e *iff* t is not an i-variable and occurs as an i-leaf of e. We say that a term t is an i-leaf if it occurs as an i-leaf in itself (i.e. it is a variable or a j-term, where $i \neq j$).

3.2.2 Expression Attributes

It is convenient to be able to annotate expressions with additional information. For this purpose, we define *expression attributes*. An expression attribute a is a mapping from the set of expressions to a set of designated values for that attribute. Implementing attributes is fairly straightforward. Thus, we simply assume that such a mechanism exists and use the notation $e.a$ to refer to the value of attribute a associated with the expression e. Two expression attributes are used by the framework.

The *find* attribute (which we will also refer to as the *find pointer*) is used to implement a standard union/find data structure for maintaining an equivalence relation on terms. The co-domain for the *find* attribute is the set of terms plus a distinguished initial value, $\perp$. The find attribute of non-terms is always $\perp$. Terms also start with their find attributes set to $\perp$, but when a term t is first used in an assertion by the framework, $t.find$ is set to t, indicating that the term is in an equivalence class by itself and is its own equivalence class representative. If two equivalence classes need to be merged, this is done by setting the find pointer of one equivalence class representative to point to the representative of the other class. At any time, the equivalence class representative of a term can be found by following a chain of find pointers until a term is reached that points to itself.

We define several abstract notions based on the *find* attribute: *hf* (short for "has find") is a predicate on expressions defined as $hf(e)$ iff $e.find \not\equiv \perp$ (we also say that e has a find pointer when $hf(e)$ holds). We define the set HF as $\{e|hf(e)\}$. $find^*$ is a partial function defined only on expressions in HF, such that $find^*(e)$ is the equivalence class representative of e. Finally, the *find* attribute induces a relation on expressions which we will denote by $\sim$: $a \sim b$ if and only if $hf(a) \wedge hf(b) \wedge find^*(a) \equiv find^*(b)$. This relation is an equivalence relation on the expressions in HF. As defined here, the relation $\sim$ is analogous to the relation $\sim$ described in algorithm *N-O*. The only difference is that the domain of $\sim$ includes all terms, not just shared terms.

The *notify* attribute (which we will also refer to as the *notify list*) is used to implement a generic call-back mechanism which is activated when equivalence classes are merged. The co-domain for the *notify* attribute is a set of pairs. The first element in each pair is a theory index i. The second is a piece of data specific to theory $\mathcal{T}_i$.

When two equivalence classes are merged, the notify list of the representative whose find pointer changes is processed. For each (i, d) in the notify list, a call is made to a theory-specific procedure $\texttt{TheoryUpdate}_i$ with d as a parameter. Notify lists are a convenient mechanism which can be used to help implement a variety of decision procedures. Initially, the notify list of each expression is $\emptyset$.

3.2.3 Global Variables

Two global variables are used by the framework: the *assertion queue* $\mathcal{Q}$ and the *inconsistent flag* $\mathcal{I}$. $\mathcal{Q}$ is a set of formulas which are waiting to be passed to `Assert`. $\mathcal{I}$ is a Boolean variable which indicates whether the current fact database has become inconsistent. It is initially `FALSE`.

3.3 The Framework

Figure 3.3 shows a diagram of the framework. As mentioned, the primary interface from the user code is `AddFact`. The user code can also call `Satisfiable`, which simply returns whether $\mathcal{I}$ is still `FALSE`. The user code can also call `Simplify` which is the simplifier mentioned earlier. The simplifier is also called as the first step of processing new formulas in `Assert`.

Of the seven theory-specific procedures, only `TheoryCheckSat`, `TheoryAssert`, and `TheoryAddSharedTerm` are essential. As mentioned above, $\texttt{TheoryCheckSat}_i$ is responsible for checking the satisfiability of the literals assigned to $\mathcal{T}_i$ (provided by `TheoryAssert`) together with the arrangement of shared terms (provided by $\texttt{TheoryAddSharedTerm}_i$) induced by $\sim$. The other theory-specific procedures are provided for convenience, and we will see examples of how they can be used below.

We now proceed to explain the framework in detail. We will begin with the framework code and then move on to the theory-specific code.

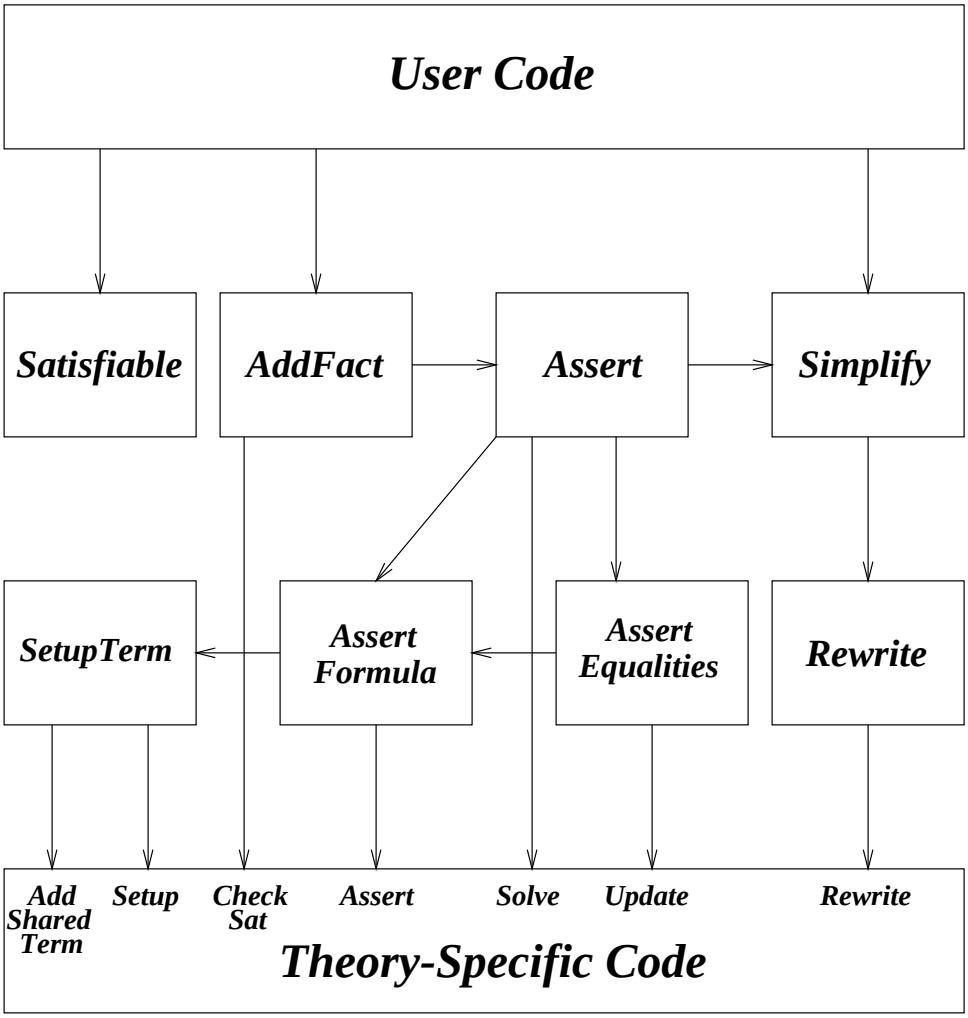

Figure 3.3: The Framework: arrows indicate caller-callee relationships

```
AddFact(e)
  Q := {e};
  REPEAT
    WHILE Q ≠ ∅ AND ¬I DO BEGIN
      Choose e* ∈ Q;
      Q := Q − {e*};
      Assert(e*);
    END
    FOR i := 1 TO N DO
      IF Q = ∅ AND ¬I THEN TheoryCheckSat_i();
  UNTIL Q = ∅ OR I;

Assert(e)
  e* := Simplify(e);
  IF Op(e*) = '=' THEN AssertEqualities(TheorySolve(e*));
  ELSE IF e* ≡ false THEN I := TRUE;
  ELSE IF e* ≢ true THEN AssertFormula(e*);

AssertEqualities(E)
  IF false ∈ E THEN I := TRUE;
  ELSE BEGIN
    FOREACH e ∈ E DO AssertFormula(e);
    FOREACH e ∈ E DO e[1].find := e[2];
    FOREACH e ∈ E DO FOREACH (i,d) ∈ e[1].notify DO TheoryUpdate_i(e,d);
  END

AssertFormula(e)
  FOREACH maximal sub-term t of e DO SetupTerm(t,T(e));
  TheoryAssert_T(e)(e);

SetupTerm(t,i)
  IF T(t) ≠ i THEN BEGIN
    TheoryAddSharedTerm_T(t)(t);
    TheoryAddSharedTerm_i(t);
  END
  IF HasFind(t) THEN RETURN;
  FOREACH child c of t DO SetupTerm(c,T(t));
  t.find := t;
  TheorySetup_T(t)(t);
```

Figure 3.4: Basic Framework

```
Simplify(e)
  IF HasFind(e) THEN RETURN Find(e);
  Replace each child c of e with Simplify(c);
  RETURN Rewrite(e);

Rewrite(e)
  IF HasFind(e) THEN RETURN Find(e);
  e* := OpRewrite(e);
  IF e ≢ e* THEN e* := Rewrite(e*);
  RETURN e*;

OpRewrite(e)
  IF Op(e) = '¬' THEN RETURN RewriteNegation(e);
  IF Op(e) = '=' AND e[1] ≡ e[2] THEN RETURN true;
  RETURN TheoryRewrite_{T(e)}(e);

RewriteNegation(e)
  IF e[1] ≡ true THEN RETURN false;
  IF e[1] ≡ false THEN RETURN true;
  IF Op(e[1]) ≡ '¬' THEN RETURN e[1][1];
  RETURN e;

HasFind(e)
  RETURN e.find ≢⊥;

Find(t)
  IF t.find ≡ t THEN RETURN t;
  ELSE RETURN Find(t.find);

Satisfiable()
  RETURN ¬I;
```

Figure 3.5: Basic Framework, continued

3.3.1 The Framework Code

Figures 3.4 and 3.5 show pseudocode for the basic framework. We will briefly explain each procedure.

AddFact

Formulas are incrementally processed from the assertion queue $\mathcal{Q}$. Initially, $\mathcal{Q}$ is set to contain only the new formula passed as a parameter to `AddFact`. Formulas are taken one by one from $\mathcal{Q}$ and passed to `Assert` which adds the formula to the fact database Φ. As in the previous chapter, the fact database Φ is composed of sets of formulas Φ_i, each of which is associated with a single theory $\mathcal{T}_i$ (the formal definition of Φ and Φ_i in the context of the framework is given in the description of `AssertFormula` below). Before `Assert` returns, new formulas to process may be added to $\mathcal{Q}$, or an inconsistency may be detected, in which case $\mathcal{I}$ is set to `TRUE`.

After all the formulas in the assertion queue $\mathcal{Q}$ have been processed, each theory is queried by calling `TheoryCheckSat`, a theory-specific procedure which checks if the set of formulas allocated to that theory are consistent. As with `Assert`, `TheoryCheckSat` may add new formulas to $\mathcal{Q}$ or set $\mathcal{I}$ to `FALSE`. `AddFact` loops until there are no new formulas to process or an inconsistency is detected.

Assert

`Assert` basically preprocesses formulas which are to be added to the fact database. It first calls `Simplify` which returns a simplified formula which is equivalent modulo the current fact database. Then, if the formula e provided to `Assert` is an equation, it calls `TheorySolve` (which must return an equisatisfiable set of equations) and passes the result to `AssertEqualities`. If e is a constant formula, it is handled directly: if *false*, $\mathcal{I}$ is set to `TRUE` indicating that the fact database has become inconsistent; if *true*, nothing needs to be done. Finally, if e is neither an equation nor a constant formula, it is passed to `AssertFormula`.

The framework as shown in Figures 3.4 and 3.5 assumes that arguments of `Assert` are always literals. This assumption will be true as long as only literals are passed to

`AddFact` by the user code and only literals are added to $\mathcal{Q}$ by the theory-specific code. For additional flexibility, an extension to the framework which handles non-literals is discussed below in Section 3.5.1.

AssertEqualities

`TheorySolve` may return $\{false\}$ if it determines that the formula passed to it is inconsistent. Thus, `AssertEqualities` first checks for this case and sets $\mathcal{I}$ to `TRUE` if it occurs. Otherwise, $\mathcal{E}$ is a set of equations. Each equation is sent to `AssertFormula`. Then, the *find* attribute of each left-hand side is set to its corresponding right-hand side. This ensures that $e[1] \sim e[2]$ for each $e \in \mathcal{E}$. Finally, the notify list of each left-hand side is traversed, and an appropriate call to `TheoryUpdate` is made for each entry on the list. We will see how this mechanism is used in Section 3.4.3 below.

AssertFormula

`AssertFormula` is called to officially add a formula to the fact database Φ. It first passes the maximal sub-terms of its argument on to `SetupTerm`, which then traverses them recursively. This ensures that `SetupTerm` has been called on every term in Φ, which enables the theory-specific code to maintain a number of invariants. The formula is then sent on to `TheoryAssert`. This is the point at which the formula finally is considered a part of the fact database. Formally, then, the set Φ_i is defined to be the set of all formulas which have appeared as an argument to $\texttt{TheoryAssert}_i$. The fact database Φ is defined to be the union of all the sets Φ_i. As we show formally in the appendix, the set of formulas passed to `AddFact` is satisfiable if and only if the set Φ is satisfiable if and only if $\mathcal{I}$ is `FALSE`.

SetupTerm

As just mentioned, `SetupTerm` is called on each term that appears in a formula sent to `AssertFormula`. It does several things. First of all, it identifies terms whose parent expressions are associated with a different theory: these are the shared terms.

Whenever this happens, both theories are notified of the shared term. After recursively traversing the term, `TheorySetup` is called which ensures that theory-specific code has a chance to process each term before it becomes part of the fact database. Finally, the find pointer of the term is set to point to itself. Thus, the term is added to the domain of $\sim$ and is initially in an equivalence class by itself.

Simplify

As mentioned above, `Simplify` is intended to perform quick and easy rewrites which can simplify an expression. `Simplify` traverses an expression recursively, performing rewrites at each node and rebuilding the expression from the bottom up. The main rewrite applied (and the base case for the recursion) is to replace terms that have a find pointer with their equivalence class representatives. Expressions without find pointers are rewritten via a call to `Rewrite`.

Rewrite, OpRewrite, RewriteNegation

`Rewrite` repeatedly calls an operator-specific rewrite function until the expression doesn't change (or an expression is obtained which has a find pointer in which case the equivalence class representative is returned). Thus it is important that repeated application of operator-specific rewrites eventually leave an expression unchanged. The operator-specific rewrites are handled by `OpRewrite`. Simple rewrites for equality and negation are included. Theory-specific operators are handled by a call to `TheoryRewrite`.

HasFind and Find

`HasFind` and `Find` are simple implementations of the *hf* and find^* functions described earlier. Although not shown in the code (for the sake of simplicity), a useful optimization to `Find` is *path compression* in which each expression's find pointer is set to the result of the recursive call before returning the result.

Satisfiable

After each call to `AddFact`, the fact database is consistent unless $\mathcal{I}$ has been set to `TRUE`. Thus, this function simply returns whether $\mathcal{I}$ is `FALSE`.

3.3.2 Theory-Specific Code

In general, whenever a theory-specific procedure is called, the theory-specific code can do whatever it wants as long as it does not alter the framework data structures directly. The theory-specific code may use the global variables $\mathcal{I}$ and $\mathcal{Q}$ to communicate with the framework as follows. If at any time the theory-specific code determines that there is an inconsistency in the fact database, it may set $\mathcal{I}$ to `TRUE`. Additionally, if the theory-specific code ever needs to add a new fact to the fact database, the fact can simply be added to $\mathcal{Q}$. It will then eventually be processed by the loop in `AddFact`. We now describe each of the theory-specific procedures.

TheoryAddSharedTerm

$\texttt{TheoryAddSharedTerm}_i$ notifies a theory about a shared term which it needs to consider when checking satisfiability. Formally, $\texttt{TheoryAddSharedTerm}_i$ adds its argument to a set Λ_i. The arrangement of Λ_i induced by $\sim$ is part of the set of formulas checked for satisfiability by $\texttt{TheoryCheckSat}_i$.

TheoryAssert

The purpose of $\texttt{TheoryAssert}_i$ is to notify theory $\mathcal{T}_i$ about a new formula for which it is responsible. Conceptually, $\texttt{TheoryAssert}_i$ adds its argument, an atomic i-formula to the set of formulas Φ_i maintained by theory $\mathcal{T}_i$. This set of formulas is part of the set that is checked for satisfiability in $\texttt{TheoryCheckSat}_i$.

TheoryCheckSat

$\texttt{TheoryCheckSat}_i$ must guarantee that three conditions hold after it is called. If Φ_i is the set of all formulas which have been arguments of $\texttt{TheoryAssert}_i$, Λ_i is the

set of all terms which have been arguments of $\texttt{TheoryAddSharedTerm}_i$, and $\sim_i$ is the equivalence relation on Λ_i induced by the *find* attribute (equivalently, $\sim_i$ is the equivalence relation $\sim$ defined above with its domain restricted to Λ_i), then

1. If $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i})$ is unsatisfiable, then $\mathcal{I}$ is `TRUE` (this corresponds to line 6 of algorithm *N-O*).

2. If $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ is unsatisfiable, but $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i})$ is satisfiable, then a new formula $a = b$ has been added to $\mathcal{Q}$, where $a \neq b \in D_{\sim_i}$ and $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i} \cup \{a \neq b\})$ is not satisfiable. This corresponds to line 8 of algorithm *N-O* and is always possible as long as $\mathcal{T}_i$ is convex. We will discuss the non-convex case in Section 3.5.1 below.

3. Finally, if $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ is satisfiable, then $\mathcal{I}$ and $\mathcal{Q}$ are unchanged (corresponding to the case when the condition in line 7 of algorithm *N-O* is false).

TheoryRewrite

$\texttt{TheoryRewrite}_i$ takes as input an i-expression, and returns an expression which is equivalent modulo $\mathcal{T}_i \cup \Phi_i$. For a Shostak theory $\mathcal{T}_i$, $\texttt{TheoryRewrite}_i$ can be used to implement the canonizer. $\texttt{TheoryRewrite}_i$ can often be used to improve performance. One way in which this can be done is by having $\texttt{TheoryRewrite}_i$ replace new terms by equivalent terms that have already been seen when possible, reducing the need to deduce and propagate equalities between terms that are trivially equivalent.

TheorySetup

The framework guarantees that $\texttt{TheorySetup}_{\mathcal{T}(e)}$ is called on a term t before it appears in a formula sent to the theory-specific code (via `TheoryAssert`). Typically, $\texttt{TheorySetup}_i$ takes advantage of this fact by adding appropriate call-backs to the notify list of such terms. Then, whenever a term appears as the left-hand side of an equation, the theory-specific code for the theory associated with the term will be notified. An example of how this mechanism is used is given in Section 3.4.3.

TheorySolve

Unlike other theory-specific procedures, `TheorySolve` is not parameterized by theory. Thus, at most one theory can provide an instance of `TheorySolve`. The intended use model is that if one of the theories is a Shostak theory, it provides the implementation of `TheorySolve` (see Section 3.4.3 below). Otherwise, `TheorySolve` does nothing.

TheoryUpdate

$\texttt{TheoryUpdate}_i$ is called by `AssertEqualities` for each entry in the notify list of the left-hand side of each equation. Typically, these entries are added by `TheorySetup`. Again, an example of this can be seen in Section 3.4.3.

3.3.3 Correctness of the Framework

As has been mentioned, the framework is essentially an implementation of algorithm *N-O* with a number of features to enhance performance and flexibility. Not surprisingly, then, the correctness of the framework is based on reasoning similar to that used for Theorem 2.4. In particular, if `TheoryCheckSat` fulfills the requirements listed above, then when `AddFact` terminates, either $\mathcal{I}$ is `TRUE`, indicating that the fact database is inconsistent, or the fact database is satisfiable.

This informal explanation gives some intuition as to why the framework is correct. However, part of the motivation of this work was to provide a formally verified framework. The appendix gives a detailed proof of correctness. It also lists formal preconditions and postconditions for each framework and theory-specific procedure. The proof guarantees that as long as theory-specific code respects these conditions, the result will be correct.

3.4 Using the Framework

In this section we will show how to use the framework to implement decision procedures for specific kinds and combinations of theories.

```
TheoryAddSharedTerm_i(e)
  Λ_i := Λ_i ∪ {e};

TheoryAssert_i(e)
  Φ_i := Φ_i ∪ {e};

TheoryCheckSat_i()
  RETURN;

TheoryRewrite_i(e)
  RETURN e;

TheorySetup_i(e)
  RETURN;

TheorySolve(e)
  RETURN {e};

TheoryUpdate_i(e,d)
  RETURN;
```

Figure 3.6: Default implementation of theory-specific code for theory $\mathcal{T}_i$

3.4.1 Default Implementation

Figure 3.6 presents a "default" implementation for the theory-specific code. Often, a theory only requires the functionality of a few of the theory-specific procedures. If no code for a particular theory-specific procedure is given, it should be assumed that the implementation of the other procedures is just the default implementation. Basically, the default procedures just do the bare minimum to keep the program flowing without losing information.

3.4.2 Nelson-Oppen Theories

In Section 2.2.3, we showed how the Nelson-Oppen method can be applied to combine theories which provide decision procedures for conjunctions of literals. Such theories can easily be integrated into the framework if, as in Section 2.2.3, we assume that a theory $\mathcal{T}_i$ provides a satisfiability procedure Sat_i satisfying the following specification

```
TheoryCheckSat_i()
  IF ¬Sat_i(Φ_i ∪ E_~i) THEN I := TRUE;
  ELSE IF ¬Sat_i(Φ_i ∪ Ar_~i) THEN BEGIN
    Choose a ≠ b ∈ D_~i such that ¬Sat_i(Φ_i ∪ E_~i ∪ {a ≠ b});
    Q := Q ∪ {a = b};
  END
```

Figure 3.7: Theory-specific code for a Nelson-Oppen theory $\mathcal{T}_i$

for an arbitrary set of Σ-literals Θ:

$$Sat_i(\Theta) = \texttt{TRUE} \textit{ iff } \mathcal{T}_i \cup \gamma_i(\Theta) \not\models \textit{false}.$$

Figure 3.7 shows the theory-specific code needed to integrate such a theory $\mathcal{T}_i$ into the framework. As might be expected, the implementation of $\texttt{TheoryCheckSat}_i$ is very similar to lines 6 through 11 of algorithm *N-O* and basically does exactly what is needed to guarantee that the three conditions mentioned in the description of $\texttt{TheoryCheckSat}_i$ are satisfied. Note that as in algorithm *N-O*, this implementation requires the theory to be convex. A more general implementation for non-convex theories is described in Section 3.5.1 below. The additional functionality of the other theory-specific procedures is not required. A formal justification of the correctness of this implementation is contained in the appendix.

3.4.3 Shostak Theories

Figure 3.8 shows the theory-specific code needed to implement a Shostak theory $\mathcal{T}_i$ with canonizer *canon* and solver *solve*. This implementation, while relying on the same correctness arguments as Algorithm *S2* from Section 2.3.1, is closer in spirit to Shostak's original algorithm [35, 10]. As a result, some additional requirements (essentially the same ones required by Shostak) on *canon* and *solve* are required beyond those listed in 2.1.

1. *canon* is a computable function from Σ_i-terms to Σ_i-terms, such that

   (a) $\mathcal{T}_i \models a = b$ iff $canon(a) \equiv canon(b)$

```
TheoryCheckSat_i()
  FOREACH e in Φ_i DO
    IF Op(e) = '¬' AND Find(e[1][1]) ≡ Find(e[1][2]) THEN BEGIN
      I := TRUE;
      RETURN;
    END

TheoryRewrite_i(e)
  IF e is not a term THEN RETURN e;
  e* := RewriteHelper(e);
  RETURN canon_i(e*);

TheorySolve(e)
  RETURN γ_i^{-1}(solve(γ_i(e)));

TheorySetup_i(e)
  IF e is a compound i-term THEN
    FOREACH c which occurs as an i-leaf in e DO
      c.notify := c.notify ∪ {(i, e)};

TheoryUpdate_i(e,d)
  IF ¬I AND Find(d) ≡ d THEN BEGIN
    d* := TheoryRewrite_i(d);
    AssertEqualities({d = d*});
  END

RewriteHelper(e)
  IF e is an i-leaf THEN BEGIN
    IF ¬HasFind(e) OR e.find ≡ e THEN RETURN e;
    ELSE RETURN RewriteHelper(Find(e));
  END ELSE BEGIN
    Replace each child c of e with RewriteHelper(c);
    RETURN e;
  END
```

Figure 3.8: Code for Implementing a Shostak theory $\mathcal{T}_i$

(b) $canon(canon(t)) \equiv canon(t)$ for all terms t.

(c) $canon(t)$ contains only variables occurring in t.

(d) $canon(t) \equiv t$ if t is a variable.

(e) If $canon(t)$ is a compound term, then $canon(c) \equiv c$ for each child c of $canon(t)$.

2. *solve* is a computable function from Σ_i-equations to sets of Σ_i-formulas defined as follows:

(a) If $\mathcal{T}_i \models x \neq y$ then $solve(x = y) \equiv \{false\}$.

(b) If $\mathcal{T}_i \models x = y$ then $solve(x = y) \equiv \emptyset$.

(c) Otherwise, $solve(x = y)$ returns a set of equations $\mathcal{E}$ in solved form such that $\mathcal{T} \models [(a = b) \leftrightarrow \exists\, \overline{w}.\, \mathcal{E}]$, where $\overline{w}$ is the set of variables which appear in $\mathcal{E}$ but not in a or b. Each of these variables must be fresh. We also require that for each $s = t \in \mathcal{E}$, $canon(t) \equiv t$.

As in Section 2.3.1, we must use a slightly modified version of *canon* and *solve* in a multi-theory environment. Thus, let $canon_i(\alpha)$ denote $\gamma_i^{-1}(canon(\gamma_i(\alpha)))$ and $solve_i(\beta)$ denote $\gamma_i^{-1}(solve(\gamma_i(\beta)))$. We further define the *Shostak database* to be a set $\mathcal{S}$ of equations in i-solved form as follows. Initially $\mathcal{S}$ is empty. Then, every time `AssertEqualities`$(\mathcal{E})$ is called from `Assert`, $\mathcal{S}$ is updated to be $\mathcal{E}(\mathcal{S}) \cup \mathcal{E}$. This requires that $\mathcal{E}$ is in solved form and $\mathcal{S}(\mathcal{E}) = \mathcal{E}$ whenever `AssertEqualities` is called from `Assert`. Fortunately, these conditions are guaranteed by the call to `TheorySolve` which precedes the call to `AssertEqualities` (see the appendix for details).

The correctness of the implementation in Figure 3.8 relies on the use of a clever technique used by Shostak's original algorithm. Specifically, the following invariant is maintained for all terms $t \in HF$:

$$\texttt{Find}(t) \equiv canon_i(\mathcal{S}(t))$$

Given this invariant, the implementation of `TheoryCheckSat` simply needs to check whether there is any disequality $a \neq b$ in Φ_i where $\texttt{Find}(a) \equiv \texttt{Find}(b)$. To see why, first consider the following proposition.

Proposition 3.1. *Suppose $\mathcal{S}$ is a set of equations in i-solved form. If for every term $t \in HF$, $\texttt{Find}(t) \equiv canon_i(\mathcal{S}(t))$, then for arbitrary terms s,$t \in HF$, $s \sim t$ if and only if $\mathcal{T}_i \cup \gamma_i(\mathcal{S}) \models \gamma_i(s = t)$.*

Proof.

$s \sim t$	$\Leftrightarrow$	$\texttt{Find}(s) \equiv \texttt{Find}(t)$	def. of $\sim$
	$\Leftrightarrow$	$canon_i(\mathcal{S}(s)) \equiv canon_i(\mathcal{S}(t))$	hypothesis
	$\Leftrightarrow$	$\gamma_i^{-1}(canon(\gamma_i(\mathcal{S}(s)))) \equiv \gamma_i^{-1}(canon(\gamma_i(\mathcal{S}(t))))$	def. of $canon_i$
	$\Leftrightarrow$	$canon(\gamma_i(\mathcal{S}(s))) \equiv canon(\gamma_i(\mathcal{S}(t)))$	def. of γ_i^{-1}
	$\Leftrightarrow$	$\mathcal{T}_i \models \gamma_i(\mathcal{S}(s)) = \gamma_i(\mathcal{S}(t))$	def. of $canon$
	$\Leftrightarrow$	$\mathcal{T}_i \models [\gamma_i(\mathcal{S})](\gamma_i(s = t))$	def. of i-solved form
	$\Leftrightarrow$	$\mathcal{T}_i \cup \gamma_i(\mathcal{S}) \models \gamma_i(s = t)$	Prop. 2.1

□

Now, consider a call to the theory-specific procedure $\texttt{TheoryCheckSat}_i$. It must determine whether $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ is satisfiable. Consider an equation $s = t \in \Phi_i \cup Ar_{\sim_i}$. The framework guarantees that any such equation must have the property that $s \sim t$ (this is by definition when $s = t \in Ar_{\sim_i}$; for the case when $s = t \in \Phi_i$, see the justification of property S_3 in the appendix). Thus, by the above proposition, $\mathcal{T}_i \cup \gamma_i(\mathcal{S}) \models \gamma_i(s = t)$. Now consider a disequation $s \neq t \in \Phi_i \cup Ar_{\sim_i}$. After the completion of $\texttt{TheoryCheckSat}_i$, if $\mathcal{I}$ has not been set, we know that $\texttt{s} \not\sim t$, and thus $\mathcal{T}_i \cup \gamma_i(\mathcal{S}) \not\models \gamma_i(s = t)$. It follows by convexity that since $\mathcal{T}_i \cup \gamma_i(\mathcal{S})$ is satisfiable (by Corollary 2.1), $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ must also be satisfiable. More details on this correctness argument can be found in the appendix.

As just shown, the correctness of `TheoryCheckSat` relies on the fact that for each term $t \in HF$, $\texttt{Find}(t) \equiv canon_i(\mathcal{S}(t))$. In order to maintain this invariant, the `TheoryUpdate` mechanism is used as follows: when an equation $a = b$ is processed by `AssertEqualities`, $a = b$ gets added to $\mathcal{S}$. This means that a is now a solitary variable of $\mathcal{S}$ and any term t in which a occurs as an i-leaf has a new value for

$canon_i(\mathcal{S}(t))$. These terms are updated while traversing the notify list of a. To ensure that this happens, `TheorySetup` puts a call-back on the notify list of each term that occurs as an i-leaf in a compound i-term.

`TheoryRewrite`(e) simply calculates $canon_i(\mathcal{S}(e))$. However, because $\mathcal{S}$ is not explicitly represented, a helper function is required. Essentially, this helper function recursively traverses e replacing solitary variables of $\mathcal{S}$ with their corresponding right-hand sides. After calling the canonizer, the result is guaranteed to be equal to $canon_i(\mathcal{S}(e))$.

Again, a formal justification of the correctness of this implementation is contained in the appendix.

3.5 Extensions to the Framework

In this section, we discuss two extensions to the framework just presented. The first is an extension which allows the framework to handle non-convex theories.

3.5.1 Non-convex Theories

The main difficulty with handling non-convex theories is that it may be impossible to meet the conditions of `TheoryCheckSat`. Recall that $Ar_{\sim_i}$ is the arrangement of terms in Λ_i induced by the $\sim$ relation and that $E_{\sim_i}$ is the set of equations in $Ar_{\sim_i}$ and $D_{\sim_i}$ is the set of disequations in $Ar_{\sim_i}$. Consider a call to `TheoryCheckSat`$_i$ where $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ is unsatisfiable but $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i})$ is satisfiable. In other words, the addition of $D_{\sim_i}$ to $E_{\sim_i}$ causes an inconsistency. Thus, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i}) \models \neg D_{\sim_i}$. Now, since $\neg D_{\sim_i}$ is a disjunction of equalities, it is possible to choose a single equality $a = b$ from $\neg D_{\sim_i}$ such that $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i}) \models a = b$ when $\mathcal{T}_i$ is convex. This equality can then be added to $\mathcal{Q}$ fulfilling the requirements for `TheoryCheckSat` described in Section 3.3.2.

If the theory is not convex, it may not be possible to find a single equality in $\neg D_{\sim_i}$ that fulfills the requirement. The solution is to relax the requirement that only literals be added to $\mathcal{Q}$. The new requirement is that some expression e be added to

$\mathcal{Q}$ where $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i}) \models e$ and $\neg D_{\sim_i} \rightarrow e$. In particular, choosing $e \equiv \neg D_{\sim_i}$ will always work.

In order to extend the framework to handle non-literals, we add a new global state variable: a set of non-literals, $\mathcal{N}$, which is initially $\emptyset$. Whenever a non-literal is asserted, it is added to this set of formulas. We define the predicate *convex* to be true if and only if for each atomic formula ϕ which is a sub-expression of some formula in $\mathcal{N}$, either $\phi \in \Phi$ or $\neg\phi \in \Phi$. The framework is only guaranteed to give the right answer if *convex* is true. The changes required by the framework as well as the theory-specific code for a non-convex Nelson-Oppen style theory are shown in Figure 3.9.

Unfortunately, the result of these changes is that the framework does not know if the fact database is consistent unless *convex* holds. Ensuring that *convex* is true whenever the consistency needs to be checked becomes the responsibility of the user code.

One way to do this is to change the code for `Satisfiable` to recursively split on the atomic formulas in $\mathcal{N}$ as follows.

```
Satisfiable()
  IF convex THEN RETURN ¬ℐ;
  Let φ be an atomic formula in 𝒩 such that φ ∉ Φ and ¬φ ∉ Φ;
  h := Save();
  AddFact(φ);
  sat := Satisfiable();
  Restore(h);
  IF sat THEN BEGIN
    AddFact(¬φ);
    sat := Satisfiable();
  END
  Restore(h);
  RETURN sat;
```

Notice, however, that this code is very similar to that of the case-splitting tactic introduced in Section 1.3. This suggests an alternate way to deal with the formulas in $\mathcal{N}$. Rather than changing the code for `Satisfiable`, we can instead change the

```
AddFact(e)
  Q := {e};
  REPEAT
    WHILE Q ≠ ∅ AND ¬I DO BEGIN
      Choose e* ∈ Q;
      Q := Q − {e*};
      Assert(e*);
    END
    FOR i := 1 TO N DO
      IF Q = ∅ AND ¬I AND convex THEN TheoryCheckSat_i();
  UNTIL Q = ∅ OR I;

Assert(e)
  e* := Simplify(e);
  IF e* is not a literal THEN N := N ∪ {e*};
  ELSE IF Op(e*) = '=' THEN AssertEqualities(TheorySolve(e*));
  ELSE IF e* ≡ false THEN I := TRUE;
  ELSE IF e* ≢ true THEN AssertFormula(e*);

OpRewrite(e)
  IF Op(e) = '¬' THEN RETURN RewriteNegation(e);
  IF Op(e) = '=' AND e[1] ≡ e[2] THEN RETURN true;
  IF e is a term or an atomic formula THEN RETURN TheoryRewrite_T(e)(e);
  RETURN e;

TheoryCheckSat_i()
  IF ¬Sat_i(Φ_i ∪ E_~i) THEN I := TRUE;
  ELSE IF ¬Sat_i(Φ_i ∪ Ar_~i) THEN BEGIN
    Choose Δ ⊆ D_~i such that ¬Sat_i(Φ_i ∪ E_~i ∪ Δ);
    Q := {¬Δ};
  END
```

Figure 3.9: Extensions for Handling Non-Convex Theories

code for ApplyTactic as follows:

```
ApplyTactic(h, c)
  Restore(h);
  IF ¬convex THEN BEGIN
    c := c ∨ ¬𝒩;
    𝒩 := ∅;
  END
  Let φ be an atomic formula appearing in c;
  AddFact(φ);
  c1 := Simplify(c);
  h1 := Save();
  Restore(h);
  AddFact(¬φ);
  c2 := Simplify(c);
  h2 := Save();
  RETURN (h1, c1), (h2, c2);
```

This modification simply removes non-literals and adds them as part of the current goal formula. Essentially, it implements a simple inference rule for moving formulas from the hypothesis to the conclusion of a sequent. The advantage of this approach is that there is only one piece of code choosing formulas on which to perform case splits. This can be very important since it is often necessary to use a sophisticated strategy to choose among all possible case splits, and a wrong choice can lead to significant degradation in performance. We discuss the problem of choosing case splits and one possible solution in the next chapter.

3.5.2 Allowing Theories to Introduce Fresh Variables

A second, and more subtle extension to the framework involves allowing theory-specific code to introduce fresh variables as long as the fact database is equisatisfiable.

Since this extension was motivated by a specific example, we will describe it. One of the theories implemented in CVC is a theory of infinite arrays [38].

Now suppose that Φ is a set of literals in this theory. If Φ contains a disequality between two arrays, it is convenient to replace it with a disequality between elements of the arrays. For example, if a and b are array terms, then the literal $a \neq b$ can

be replaced by $a[k] \neq b[k]$, where k is a fresh variable. The result is a set of literals equisatisfiable with Φ, though not logically equivalent to Φ.

In order to support such techniques, we allow a theory $\mathcal{T}_i$ to introduce a new formula ϕ into Φ_i as long as $\mathcal{T}_i \cup \gamma_i(\Phi_i) \models \exists \overline{w}.\, \gamma_i(\phi)$, where $\overline{w}$ are fresh variables and $\overline{w} = \mathit{free}(\phi) - \mathit{free}(\Phi_i)$.

The appendix contains a proof of correctness for the framework code introduced in this chapter, including the extensions described in this section.

Chapter 4

Incremental Translation to SAT

In the past few years, general-purpose propositional satisfiability (SAT) solvers have improved dramatically in performance and have been used to tackle many new problems. It has also been shown that certain simple fragments of first-order logic can be decided efficiently by first translating the problem into an equivalent SAT problem and then using a fast SAT solver. By using appropriate tricks to reduce the time and space required for the translation, this approach seems to work well for simple theories such as the theory of equality with uninterpreted functions [7, 33]. However, it is not clear how or whether such an approach would work for other decidable theories.

In this chapter, we propose a method designed to be more generally applicable: given a satisfiability procedure Sat_{FO} (like that described in previous chapters) for a conjunction of literals in some first-order theory, a fast SAT-based satisfiability procedure for *arbitrary* quantifier-free formulas of the theory can be constructed by abstracting the formula to a propositional approximation and then incrementally refining the approximation until a sufficiently precise approximation is obtained to solve the problem. The refinement is accomplished by using Sat_{FO} to diagnose conflicts and then adding the appropriate *conflict clauses* to the propositional approximation.

We begin with a brief review of propositional satisfiability. We then describe the problem in Section 4.2. Section 4.3 describes our approach to solving the problem using SAT, and Section 4.4 describes a number of difficulties that had to be overcome in order to make the approach practical. Section 4.5 describes some related work,

```
propositional formula ::= true | false | propositional variable
      | propositional formula ∧ propositional formula
      | propositional formula ∨ propositional formula
      | ¬propositional formula

CNF formula ::= (clause ∧ ... ∧ clause)

clause ::= (propositional literal ∨ ... ∨ propositional literal)

propositional literal ::= propositional variable
      | ¬propositional variable
```

Figure 4.1: Propositional logic and CNF

and in Section 4.6, we give results obtained using CVC [37], a new decision procedure for a combination of theories in a quantifier-free fragment of first-order logic which includes the SAT solver Chaff [28]. We compare with results using CVC without Chaff and with our best previous results using SVC [1], the predecessor to CVC. The new method is generally faster, requires significantly fewer decisions, and is able to solve examples which were previously too difficult.

4.1 Propositional Satisfiability

The SAT problem is the original classic NP-complete problem of computer science. A propositional formula is built as shown in Fig. 4.1 from propositional variables (i.e. variables that can either be assigned *true* or *false*) and Boolean operators ($\wedge$, $\vee$, $\neg$). Given such a formula, the goal of SAT is to find an assignment of *true* or *false* to each variable which results in the entire formula being *true*.

Instances of the SAT problem are typically given in conjunctive normal form (CNF). As shown in Fig. 4.1, CNF requires that the formula be a conjunction of *clauses*, each of which is a disjunction of propositional literals. In Section 4.3.1, we describe a well-known technique for transforming any propositional formula into an equisatisfiable propositional formula in conjunctive normal form.

Although the SAT problem is NP-complete, a wide variety of techniques have been developed that enable many examples to be solved very quickly. A large number of

```
formula ::= true | false | literal
      | term = term
      | predicate symbol(term, ..., term)
      | formula ∧ formula
      | formula ∨ formula
      | ¬formula

literal ::= atomic formula | ¬atomic formula

atomic formula ::= atomic term = atomic term
      | predicate symbol(atomic term, ..., atomic term)

term ::= atomic term
      | function symbol(term, ...,term)
      | ite(formula,term,term)

atomic term  ::= variable | constant symbol
      | function symbol(atomic term, ...,atomic term)
```

Figure 4.2: A quantifier-free fragment of first-order logic

publicly distributed algorithms and benchmarks are available [36]. Chaff [28] is a SAT solver developed at Princeton University. As with most other SAT solvers, it requires that its input be in CNF. It is widely regarded as one of the best performing SAT solvers currently available.

4.2 The Problem

We will show how to use SAT to aid in determining the satisfiability of a formula ϕ in a language which is much more expressive than propositional logic: the basic variant of quantifier-free first-order logic shown in Fig. 4.2. Note that in the remainder of this chapter, the term "literal" by itself will be used to refer to an atomic formula or its negation, as defined in Fig. 4.2. This differs from the term "propositional literal" which we will use as in the previous section to mean a propositional variable or its negation. A small difference between this logic and conventional first-order logic is the inclusion of the *ite* (*if-then-else*) operator which makes it possible to compactly represent a term which may have one of two values depending on a Boolean condition,

a situation which is common in applications. An *ite* expression contains a formula and two terms. The semantics are that if the formula is true, then the value of the expression is the first term, otherwise the value of the expression is the second term. Note that while both formulas and terms may contain proper Boolean sub-expressions, *atomic* formulas and *atomic* terms do not.

In previous chapters, we described a fast algorithm for determining the satisfiability of conjunctions of literals with respect to some logical theory (or combination of theories). We do not address the issue of constructing such decision procedures here, but rather assume that we are given a decision procedure Sat_{FO} for determining the satisfiability, with respect to a theory of interest, of a conjunction of literals in the logic of Fig. 2.

The problem we will address is how to use such a decision procedure to construct an efficient SAT-based decision procedure for the satisfiability of arbitrary formulas (i.e. not just conjunctions of literals).

4.3 Checking Satisfiability of Arbitrary Formulas using SAT

Suppose we have, as stated, a core decision procedure Sat_{FO} for determining the satisfiability of conjunctions of literals, and we wish to determine whether an arbitrary formula ϕ is satisfiable.

An obvious approach would be to use propositional transformations (such as distributivity and DeMorgan's laws) to transform ϕ into a logically equivalent disjunction of conjunctions of literals and then test each conjunct for satisfiability using Sat_{FO}. Unfortunately, this transformation can increase the size of the formula exponentially, and is thus too costly in practice.

As mentioned in the first chapter, the approach taken by CVC consists of a high-level proof search built on top of a core decision procedure for satisfiability. In Fig. 1.3 (repeated here as Fig. 4.3 for convenience), an atomic formula is chosen from the conclusion c, but no guidance is given on how to choose that formula. It turns out

```
ApplyTactic(h, c)
  Let φ be an atomic formula appearing in c;
  h1 := AddFact(h, φ);
  c1 := Simplify(h1, c);
  h2 := AddFact(h, ¬φ);
  c2 := Simplify(h2, c);
  RETURN (h1, c1), (h2, c2);
```

Figure 4.3: Case-Splitting Tactic

that the order in which atomic formulas are chosen can affect the performance of the algorithm by several orders of magnitude.

In previous work on SVC [1, 27], various heuristics were developed for choosing which formulas to split on. Though powerful and effective on many examples, they were also somewhat *ad hoc* and not very robust: small changes or differences in formulas can cause a dramatic change in the number of decisions made and the amount of time taken.

The new approach described in this chapter is designed to be fast and robust. The key idea is to incrementally form a propositional abstraction of a first-order formula. Consider an abstraction function Abs which maps first-order formulas to propositional formulas. It is desirable that the abstraction have the following two properties:

1. For any formula ϕ, if $Abs(\phi)$ is unsatisfiable, then ϕ is unsatisfiable.

2. If $Abs(\phi)$ is satisfiable, then the abstract solution can either be translated back into a solution for ϕ or be used to refine the abstraction.

We first describe a process for determining an appropriate initial propositional abstraction Abs. We then describe how to refine the abstraction if the proof attempt is inconclusive.

4.3.1 Computing an Abstraction Formula

The basic idea of the process is to replace non-propositional formulas with propositional variables. Each syntactically distinct atomic formula α is replaced with a fresh

propositional variable, p_α. Syntactically identical atomic formulas are replaced with the same propositional variable.

The result would be a purely propositional formula if not for the *ite* operator. Handling this operator requires a bit more work. We use a transformation which preserves satisfiability and eliminates the *ite* expressions. First, each *ite* term t is replaced with a fresh term variable v_t. Again, syntactically identical terms are replaced with the same variable. Then for each syntactically distinct term $t \equiv ite(a, b, c)$ that is replaced, the following formula is conjoined to the original formula: $(a \rightarrow v_t = b) \wedge (\neg a \rightarrow v_t = c)$. By repeating this process, all *ite* operators can be eliminated (in linear time), and in the resulting formula, all terms are atomic. Atomic formulas can then be replaced by propositional variables, as described above, and the resulting formula is purely propositional.

To convert the resulting propositional formula to CNF in linear time, we employ a standard technique [26]: a fresh propositional variable is introduced for each syntactically distinct non-variable sub-formula. Then, a set of CNF clauses is produced for each sub-formula which describes the relationship of the formula to its children. The translations for each of the standard Boolean operators are as follows.

$$\begin{aligned} a := \neg b \quad &\longrightarrow \quad (a \vee b) \wedge (\neg a \vee \neg b) \\ a := b \wedge c \quad &\longrightarrow \quad (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b) \wedge (\neg a \vee c) \\ a := b \vee c \quad &\longrightarrow \quad (\neg a \vee b \vee c) \wedge (a \vee \neg b) \wedge (a \vee \neg c) \end{aligned}$$

Now, suppose that $Abs(\phi)$ is satisfiable and that the solution is given as a conjunction ψ of propositional literals. This solution can be converted into an equivalent first-order solution by inverting the abstraction mapping on the solution (replacing each propositional variable p_α in ψ with α). Call the result $Abs^{-1}(\psi)$. Since $Abs^{-1}(\psi)$ is a conjunction of literals, its satisfiability can be determined using Sat_{FO}. If $Abs^{-1}(\psi)$ is satisfiable, then in the interpretation which satisfies it, the original formula ϕ must reduce to *true*, and thus ϕ is satisfiable. Otherwise, the result of the experiment is inconclusive, meaning that the abstraction must be refined. We describe how to do this next.

4.3.2 Refining the Abstraction

An obvious approach to refining the abstraction is to add a clause to the propositional formula that rules out the solution determined to be invalid by Sat_{FO}. Since ψ is a conjunction of propositional literals, applying de Morgan's law to $\neg\psi$ yields a standard propositional clause. Thus, $Abs(\phi) \wedge \neg\psi$ is a refinement of the original abstraction which rules out the invalid solution ψ. Furthermore, the refinement is still in CNF as required. We call the clause $\neg\psi$ a a *conflict clause* because it captures a set of propositional literals which conflict, causing an inconsistency. This is in accordance with standard SAT terminology. However, in standard SAT algorithms, conflict clauses are obtained by analyzing a clause which has become false to see which decisions made by the SAT solver are responsible. In our approach, the conflict clause is obtained by an agent outside of the SAT solver. After refining the abstraction by adding a conflict clause, the SAT algorithm can be restarted. By repeating this process, the abstraction will hopefully be refined enough so that it can either be proved unsatisfiable by the SAT solver or the solution ψ provided by SAT can be shown to map to a satisfying assignment for the original formula.

4.4 The Difficult Path to Success

There are a surprising number of roadblocks on the way from the previous idea to a practical algorithm. In this section we describe some of these and our solutions.

4.4.1 Redundant Clauses

The most severe problem with the naive approach outlined above is that it tends to produce an enormous number of redundant clauses. To see why, suppose that SAT computes a solution consisting of $n + 2$ propositional literals, but that only the last two propositional literals contribute to the inconsistency of the equivalent first-order set of literals. Then, for each assignment of values to the other n propositional variables which leads to a satisfying solution, the refinement loop will have to add another clause. In the worst case, the refinement loop will have to add 2^n clauses.

This is particularly troubling because a single clause, one containing just the two contributing propositional literals would suffice.

In order to avoid the problem just described, the refinement must be more precise. In particular, when Sat_{FO} is given a set of literals to check for consistency, an effort must be made to find the *smallest* possible subset of the given set which is still inconsistent. Then, a clause derived from *only these* literals can be added to the propositional formula.

One possible way to implement this is to minimize the solution by trial and error: starting with n literals, pick one of the literals and remove it from the set. If the set is still inconsistent, leave that literal out; otherwise, return it to the set. Continue with each of the other literals. At the end, the set will contain a minimal set of literals. Unfortunately, this approach requires having Sat_{FO} process $O(n)$ literals n times for each iteration of the refinement loop (where n is the number of variables in the abstract formula). A few experiments with this approach quickly reveal that it is far too costly to give a practical algorithm.

A more practical solution, though one which is not trivial to implement, is to have the decision procedure Sat_{FO} maintain enough information to be able to report directly which subset of a set of inconsistent literals is responsible for the inconsistency.

Fortunately, through a discussion with Cormac Flanagan [18], we realized that this is not difficult to do in CVC. This is because CVC is a proof-producing decision procedure, meaning that it is possible to have CVC generate an actual proof of any fact that it can prove. Using the infrastructure for proof production in CVC, we implemented a mechanism for generating *abstract proofs*. In abstract proof mode, CVC just tracks the external assumptions that are required for each proof. The result is that when a set of literals is reported by CVC to be inconsistent, the abstract proof of inconsistency contains exactly the *subset* of those literals that would be used to generate a proof of the inconsistency. The abstract proof thus provides a subset which is known to be inconsistent. This subset is not guaranteed to be minimal, but we found that in most cases it is very close to minimal. Since the overhead required to keep track of abstract proofs is small (typically around 20%), abstract proofs provide an efficient and practical solution for eliminating the problem of redundant clauses.

4.4.2 Lazy vs. Eager Notification

The approach described in the previous section is *lazy* (see the note in Section 4.5 below) in the sense that the SAT solver is used as a black box and the first-order procedure Sat_{FO} is not invoked until a solution is obtained from the SAT solver. Unfortunately, as shown in Table 4.3, the lazy approach becomes impractical for problems which require many refinements. In contrast, an *eager* approach is to notify the first-order procedure Sat_{FO} of every decision that is made (or unmade) by the SAT solver. Then, if an inconsistency is detected by Sat_{FO}, it is immediately diagnosed, providing a new conflict clause for SAT. The SAT algorithm then continues, never having to be restarted.

The performance advantages of the eager approach are significant. The disadvantages are that it requires more functionality of both the SAT solver and the decision procedure Sat_{FO}. The SAT solver is required to give notification every time it makes (or revokes) a decision. Furthermore, it must be able to accept new clauses in the middle of solving a problem (CVC includes a modified version of Chaff which has this functionality). The eager approach also requires Sat_{FO} to be *online*: able quickly to determine the consistency of incrementally more or fewer literals. Fortunately, CVC has this property.

4.4.3 Decision Heuristics

The decision heuristics used by Chaff and other SAT solvers consider every variable a possible target when choosing a new variable to do a case split on. However, in the abstracted first-order formula, not all variables are created equally. For example, consider an *ite* expression: $ite(\alpha, t_1, t_2)$, and suppose that t_1 and t_2 are both large non-atomic terms. If the propositional variable associated with α is set to *true*, then all of the clauses generated by the translation of t_2 can be ignored since they can no longer affect the value of the original formula. Unfortunately, the SAT solver doesn't have this information, and as a result it can waste a lot of time choosing irrelevant variables. This problem has been addressed by others [15], and our solution is similar. We annotate the propositional variables with information about the structure of the

original formula (i.e. parent/child relationships). Then, rather than invoking the built-in heuristic for variable selection, a depth-first search (DFS) is performed on the portion of the original formula which is relevant. The first variable corresponding to an atomic formula which is not already assigned a value is chosen. Although this can result in sub-optimal variable orders in some cases, it avoids the problem of splitting on irrelevant variables. Table 4.4 compares results obtained using the built-in Chaff decision heuristic with those obtained using the DFS heuristic. These are discussed in Section 4.6.

4.4.4 SAT Heuristics and Completeness

A somewhat surprising observation is that some heuristics used by SAT solvers must be disabled or the method will be incomplete. An example of this is the "pure literal" rule. This rule looks for propositional variables which have the property that only one of their two possible propositional literals appears in the formula being checked for satisfiability. When this happens, all instances of the propositional literal in question can immediately be replaced with *true*, since if a solution exists, a solution will exist in which that propositional literal is *true*.

However, if the formula is an abstraction of a first-order formula, it may be the case that a solution exists when the propositional literal is *false* even if a solution does not exist when the propositional literal is *true*. This is because the propositional literal is actually a place-holder for a first-order literal whose truth may affect the truth of other literals. Propositional literals are guaranteed to be independent of each other, while first-order literals are not. Because of this, there is no obvious way to take advantage of pure literals and the rule must be disabled. Fortunately, this was the only such rule that had to be disabled in Chaff.

4.4.5 Theory-specific Challenges

Finally, a particularly perplexing difficulty is dealing with first-order theories that need to do case splits in order to determine whether a set of literals is satisfiable. For example, consider a theory of arrays with two function symbols, *read* and *write*.

In this theory, $read(a, i)$ is a term which denotes the value of array a at index i. Similarly, the term $write(a, i, v)$ refers to an array which is identical to a everywhere except possibly at index i, where its value is v. Now, consider the following set of literals in this theory: $\{read(write(a, i, v), j) = x, x \neq v, x \neq a[i]\}$. In order for the array decision procedure to determine that such a set of literals is inconsistent, it must first do a case split on $i = j$. However, such additional case splits by the theories can cost a lot of time. Furthermore, they may not even be necessary to solve the problem. We found it difficult to find a strategy for integrating such case splits without adversely affecting performance. Instead, we preprocess the formulas so that such case split formulas become part of the original formula and are only split on when necessary. For the specific case of the array decision procedure, every instance of $read(write(a, i, v), j)$ is rewritten to $ite(i = j, v, read(a, i))$. Also, in order to increase the likelihood of being able to apply this rewrite, every instance of $read(ite(a, b, c), v)$ is rewritten to $ite(a, read(b, v), read(c, v))$. These rewrites are sufficient to obtain reasonable performance for our examples. However, we suspect that for more complicated examples, something more sophisticated may be required.

4.5 Related Work

Flanagan, Joshi, and Saxe at Compaq SRC independently developed a very similar approach to combining first-order decision procedures with SAT [19]. Their translation process is identical to ours. Furthermore, their approach to generating conflict clauses is somewhat more sophisticated than ours. However, their prototype implementation is lazy (the nomenclature of "lazy" versus "eager" is theirs). Also it only includes a very limited language and its performance is largely unknown. Unfortunately, we have not been able to compare directly with their implementation.

De Moura, Ruess, and Sorea at SRI have also developed a similar approach using their ICS decision procedure [14]. However, ICS is unable to produce minimal conflict clauses, so they use an optimized variation of the trial and error method described in Section 4.4.1 to minimize conflict clauses. Also, as with the Compaq approach, their implementation is lazy and its performance unknown. Though they do not report

execution times, they do provide their benchmarks, and our implementation using CVC with Chaff was able to solve all of them easily.

It would also be interesting to compare with the approach for solving problems in the logic of equality with uninterpreted functions by translating them (up front) to SAT problems. We made an attempt to perform direct comparisons with [33], but their benchmarks are not provided in the language of equality with uninterpreted functions, and unfortunately, it is not clear how to translate them. As a result, we were unable to run their benchmarks. We suspect that our approach would be competitive with theirs. However, since the logic is so simple, it is not clear that a more general approach like ours would be better.

4.6 Results

We implemented the approach described above in the CVC decision procedure using the Chaff SAT solver, and tested it using a suite of processor verification benchmarks. The first three benchmarks are purely propositional formulas from Miroslav Velev's super-scalar suite (http://www.ece.cmu.edu/∼mvelev). The next three are also from Velev's DLX verification efforts, but they include array and uninterpreted function operations. The rest are from our own efforts in processor verification and also include array and uninterpreted function operations.

These were run using gcc under linux on an 800MHz Pentium III with 2GB of memory. The best overall results were obtained by using an eager notification strategy and the DFS decision heuristic. Table 4.1 compares these results to results obtained by using CVC without Chaff (using the recursive algorithm of Fig. 1.2). As can be seen, the results are better, often by several orders of magnitude, in every case but one (the easiest example which is solved by both methods very quickly). These results show that CVC with Chaff is a significant improvement over CVC alone.

Our goal in integrating Chaff into CVC was not only to test the feasibility of the approach, but also to produce a tool which could compete with and improve upon the best results obtained by our previous tool, SVC. SVC uses a set of clever but somewhat *ad hoc* heuristics to improve on the performance obtained by the algorithm of Fig.

Table 4.1: Results comparing CVC without Chaff to CVC combined with Chaff

Example	CVC without Chaff		CVC+Chaff	
	Decisions	Time (s)	Decisions	Time (s)
bool-dlx1-c	?	> 10000	2522	1.14
bool-dlx2-aa	?	> 10000	792	0.81
bool-dlx2-cc-bug01	?	> 10000	573387	833
v-dlx-pc	8642456	5082	6137	6.10
v-dlx-dmem	2888268	2820	2184	3.48
v-dlx-regfile	29435	37.6	3833	6.64
dlx-pc	515	0.68	529	1.04
dlx-dmem	6031	4.50	1276	1.90
dlx-regfile	6386	5.27	2739	4.12
pp-bloaddata-a	93714	79.1	1193	1.80
pp-bloaddata	345569	338	4451	4.51
pp-dmem2	367877	338	2070	1.52

1.2 by learning which atomic formulas are best to split on [27]. Table 4.2 compares the results obtained by SVC with the results obtained by CVC with Chaff.

SVC performs particularly well on the last 6 examples, a fact which is not too surprising since these are old benchmarks that were used to tune SVC's heuristics. However, SVC's performance on the first six examples shows that it's heuristics are simply not flexible enough to handle a large variety of formulas. CVC, on the other hand produces good results fairly consistently. Even in the four cases where CVC is slower than SVC, the number of decisions is comparable, and in all other cases the number of decisions required by CVC is much less. This is encouraging because it means that CVC is finding shorter proofs, and additional performance gains can probably be obtained by tuning the code. Thus, overall, CVC seems to perform better and to be more robust than SVC, which is the goal we set out to accomplish.

Table 4.2: Results comparing SVC to CVC

Example	SVC		CVC+Chaff	
	Decisions	Time (s)	Decisions	Time (s)
bool-dlx1-c	11228452	776	2522	1.14
bool-dlx2-aa	?	> 10000	792	0.81
bool-dlx2-cc-bug01	?	> 10000	573387	833
v-dlx-pc	4620149	503	6137	6.10
v-dlx-dmem	199540	31.7	2184	3.48
v-dlx-regfile	74600	18.2	3833	6.64
dlx-pc	384	0.15	529	1.04
dlx-dmem	655	0.21	1276	1.90
dlx-regfile	936	0.27	2739	4.12
pp-bloaddata-a	902	0.66	1193	1.80
pp-bloaddata	35491	5.35	4451	4.51
pp-dmem2	47989	7.54	2070	1.52

Table 4.3: Results comparing naive, lazy, and eager implementations

Example	Naive		Lazy		Eager
	Iterations	Time (s)	Iterations	Time (s)	Time (s)
read0	77	0.14	17	0.09	0.07
pp-pc-s2i	?	> 10000	82	1.36	0.10
pp-invariant	?	> 10000	239	5.81	0.22
v-dlx-pc	?	> 10000	6158	792	3.22
v-dlx-dmem	?	> 10000	?	> 10000	4.12

4.6.1 Comparing Different Strategies

Finally, we show experimental results for some of the different strategies discussed in the previous section. First, just to drive the point home, we show a simple comparison of the naive (lazy without minimal conflict clauses), lazy (with minimal conflict clauses), and eager (with minimal conflict clauses) implementations on some simple examples. As can be seen, the naive and lazy approaches quickly become impractical.

Next, we compare two versions of the eager approach with minimal conflict clauses: one using the standard Chaff decision heuristics, and one using the DFS heuristic

Table 4.4: Variable selection by Chaff vs. by depth-first search

Example	Chaff		DFS	
	Decisions	Time (s)	Decisions	Time (s)
bool-dlx1-c	1309	0.69	2522	1.14
bool-dlx2-aa	4974	2.36	792	0.81
bool-dlx2-cc-bug01	10903	11.4	573387	833
v-dlx-pc	4387	3.22	6137	6.10
v-dlx-dmem	5221	4.12	2184	3.48
v-dlx-regfile	6802	5.85	3833	6.64
dlx-pc	39833	19.0	529	1.04
dlx-dmem	34320	18.8	1276	1.90
dlx-regfile	47822	35.5	2739	4.12
pp-bloaddata-a	8695	5.47	1193	1.80
pp-bloaddata	9016	5.56	4451	4.51
pp-dmem2	3167	2.24	2070	1.52

discussed in Section 4.4.3. The results are shown in Table 4.4. As can be seen, DFS outperforms the standard technique on all but four examples. Two of these are purely Boolean test cases, and so the DFS method wouldn't be expected to provide any advantage. For purely propositional formulas, then, (or first-order formulas that are *mostly* propositional), the standard Chaff technique is probably better. It is particularly interesting to note how badly DFS does on the example "bool-dlx2-cc-bug01". One area for future work is trying to find a way to automatically choose between or combine these two methods.

Chapter 5

Conclusions

5.1 Contributions

The main goal of this research was to address the limitations in the Stanford Validity Checker (SVC). The most fundamental limitation of SVC was that it lacked a solid theoretical foundation. This was a significant problem, manifesting itself whenever we tried to extend or modify the program.

Partly as a result of our incomplete grasp of the theory, the requirements on decision procedures for specific theories in SVC were quite restrictive and rigid. For example, every theory was required to provide a solver which would rewrite an equation into an equisatisfiable set of equations. Furthermore, this new set of equations was required to comply with a total ordering on expressions: the right-hand side of each equation was required to be simpler than the left-hand side. This ordering requirement was often very challenging to meet and led to inefficiencies and complications in the code.

Perhaps the most severe consequence of the theoretical deficiencies of SVC was that it diminished our confidence in the correctness of the tool. Since SVC was being used as a verification tool, it was important that our confidence in its correctness be high.

As mentioned, the ordering requirement was one source of complexity in SVC. Another difficulty was simply the fact that SVC had evolved significantly and had

thus outgrown its original software architecture. As is common with such systems, new features became increasingly difficult to add. For example, one desirable new feature was the ability to suspend the proof effort for a formula, check the validity of a different formula, and then return to the proof of the original formula. Because of the complexity of the state in SVC, this turned out to be impossible to do reliably.

Finally, although SVC performed well on many examples, there were also many examples for which we knew better performance was possible.

The limitations in theory, software architecture, and performance mentioned above are addressed by the work of this thesis (roughly in chapters 2, 3, and 4 respectively). The theoretical contribution of this thesis includes new variations of both the Shostak and Nelson-Oppen methods for combining procedures. These are used to show that Shostak's method is actually an instance of the Nelson-Oppen method. Part of this effort includes a new theorem relating *convex* and *stably infinite* first-order theories.

Building upon this increased theoretical understanding, this thesis proposes a new framework for combining decision procedures for individual theories. The architecture of the new framework was designed to be significantly more flexible than that used for SVC. For example, there is much more freedom when adding a new theory. There are no ordering requirements, and the theory can be implemented using a Shostak interface, a more general Nelson-Oppen interface, or its own customized interface. Probably the most significant accomplishment of this thesis is the proof provided in the appendix, showing the correctness of the framework. The proof not only provides great confidence in the correctness of the framework (which is essential for a verification tool), but also serves as documentation of the interfaces and assumptions made by the framework. This is very helpful when trying to add or optimize theory-specific code.

Finally, this thesis presents a new technique for dramatically improving the performance of cooperating decision procedures by drawing upon the conflict diagnosis techniques of Boolean satisfiability procedures. As described in chapter 4, a number of additional enhancements were required in order to turn this promising idea into a working solution, but eventually we were able to obtain results which were better than any previous effort.

The results presented in this thesis have been incorporated in a new tool called CVC (Cooperating Validity Checker), the successor to SVC. A lot of the credit for CVC goes to my colleague, Aaron Stump, who wrote most of the actual code based on the ideas and architecture proposed in this thesis. CVC has proven to be much more robust and flexible than SVC.

5.2 Some Observations on Program Verification

By far the most difficult part of this thesis was producing the proof of correctness found in the appendix. The most obvious question in my mind after completing this task is whether having a verified algorithm is really worth the enormous effort required.

Certainly, the verified algorithm is of great value in providing a solid foundation for CVC. But it's not clear whether this value alone really justifies the the years of effort required to produce the algorithm and proof. On the other hand, once we combine this value with the fact that it provides an interesting case study in program verification, perhaps the effort is more easily justified.

5.2.1 Organizing a Large Verification Effort

One of the difficulties was simply coming up with an approach for the proof effort. At first, I tried to prove correctness without using a line by line approach. Though I was able to come up with a proof, it often required temporal reasoning, i.e. if a certain event happens, then another event must have happened earlier, etc. I became convinced that this introduced too much complexity.

I reworked the proof to come up with a line-by-line proof, where the properties are listed that are true before and after each line of code. The advantage of this approach is that the proof for each line of code can be checked without referring to any other part of the code. This makes the proof easier to read as well as less prone to errors.

Still, presenting the proof is a challenge. The ideal medium for presenting such a proof would be an interactive program which would could be configured to show

more or less detail, highlight which parts of the proof are dependent on which other parts, and so forth. Perhaps this would be a good future research project.

5.2.2 Verification: the Cost and the Benefit

Probably the most important thing I gained from this effort is an appreciation of how difficult program verification is. Certainly, this was not unexpected, as program verification has long been known to be a difficult problem. While I would argue that this algorithm is uncharacteristically complex for its size, it is remarkable how much complexity can be hidden in relatively few lines of code.

Probably the single most difficult part of the proof was coming up with the appropriate loop invariants. Of course, it's well known that this is supposed to be hard, but I was able to verify it by my own experience. One reason they are hard is that they often represent a complex intermediate state between two simpler states. Property S_{16}(see appendix) is a good example of a particularly tricky loop invariant.

The good news is that producing a loop invariant really just requires understanding how and why the loop works. In fact, I found this to be true of the verification effort in general. Of course, verification requires a level of detailed understanding rarely (if ever) achieved when writing code. Often, even when I thought I knew how part of the program worked, I discovered that my understanding was incomplete. Completing the gaps in my understanding was where real effort was required. Once I had a complete understanding, producing the proof was largely a mechanical exercise.

This is not such a tragic conclusion. Verifying an algorithm forces a level of understanding that I would argue is impossible to achieve in any other way. Especially if the code is complex or the correctness of the code is essential, there is nothing as effective as verification.

Can the verification effort be made easier? I am confident that it can be. Certainly, my effort would have benefited from some sort of proof assistant which could track progress and do bookkeeping.

However, perhaps a more effective way to simplify the effort would be to write code which does not require as much effort to understand. The algorithm I verified was

written with efficiency and compactness as its primary goals. It may have been worth sacrificing some of this to make the proof more straightforward. On the other hand, much of the complexity of this algorithm is inherent, and it's probably another major research effort to understand how such an algorithm could be modified to simplify the proof.

5.3 Future Work

Hopefully, the framework presented here is general and flexible enough to be useful for a wide class of applications. However, there are certainly applications which are beyond the scope of the current framework. A significant limitation which shows up in a variety of applications is the inability to handle quantified formulas.

5.3.1 Quantified Formulas

The formulas given to CVC are assumed to be quantifier-free. CVC must then determine if the formula is valid. If a formula is valid, this means that it is true regardless of the value of any variables in the formula. Another way to look at this is to consider all variables appearing in the formula to be implicitly universally quantified. Thus, in some sense, CVC can handle a very limited set of quantified formulas: formulas in which every variable is universally quantified. However, often it is desirable to include existential or nested quantifiers. In general, inclusion of such quantifiers often renders the theory undecidable, but there are examples of theories, such as Presburger arithmetic [8], for which the validity of arbitrarily quantified formulas is still decidable.

For such theories, it is theoretically possible to extend the framework to handle existential quantifiers in some cases. A foundation for this extension is in a paper by Tinelli and Ringeissen [41], which shows that Theorem 2.2 can be extended to non-literals, even quantified formulas, as long as the set of formulas are pure in their respective theories. The question then becomes: how and under what circumstances can a formula containing quantifiers in a combination of theories be purified?

A first result is that as long as quantified variables associated with one theory do not appear as proper sub-terms of terms alien to that theory, it is possible to purify the formulas in the usual way, simply by replacing occurrences of alien terms with fresh variables (see Section 2.2.1).

A more general solution seems to require instantiating selected quantified variables using some kind of heuristic guidance. This is a promising area of future research.

5.3.2 Restrictions on the Theories

There are two main restrictions on the kinds of theories that can be accommodated by the framework presented in chapter 3. The first is that each theory must be stably infinite. The other is that the signatures of the theories be disjoint.

Although we have not found either of these requirement to be prohibitive, a natural question is whether they can be further relaxed. Tinelli and Ringeissen have some work on combining theories with non-disjoint signatures [41], but it can only be applied to a fairly narrow range of theories.

The requirement that theories be stably infinite is known to be more general than required. In fact, it is only necessary that all theories have a model of the same cardinality. Stable infiniteness is simply an easy way to guarantee this condition.

Investigating variations on relaxing these theory requirements seems to be another possible area of future research, although it may be more compelling if specific applications can be identified which require these requirements to be relaxed.

5.3.3 Performance

Though the techniques of chapter 4 make a significant improvement to the performance of CVC, there is still more to be done. First of all, CVC is young compared to SVC, so there is still more tuning that can be done to improve the implementation.

From an algorithmic point of view, there is also more that can be done. As mentioned in chapter 4, the method for choosing which variable to split on can probably be improved. It is likely that some combination of the built-in Chaff heuristic and our depth-first search heuristic could work better than either one individually.

Appendix A

Correctness of the Framework

A.1 Approach

Correctness consists of two important parts: partial correctness and termination. Partial correctness means that if the program terminates, the result is correct. Termination means that the program always eventually terminates (i.e. there is no way for it to run forever). Although both of these are important, we present only a proof of partial correctness. This is mainly because the proof of partial correctness was already so challenging that it would have been too much effort to produce an additional proof of termination. However, from a more pragmatic standpoint, we have found that termination problems are much easier to detect and deal with than partial correctness problems. And in any case, it is very useful just to be able to say that if the program terminates, the result is correct.

Partial correctness is shown by giving preconditions and postconditions for each procedure in the framework as well as each theory-specific procedure. Each procedure is then shown to guarantee its postconditions under the assumption that the preconditions hold when it is called.

Since procedures may call each other recursively, for each proof of each procedure, we assume that all other procedures are correct with respect to their preconditions and postconditions. The following argument shows that this methodology is still sufficient to guarantee partial correctness.

Proposition A.1. *Let $P_1, \ldots, P_n$ be a set of procedures which may call each other recursively. Suppose we can prove partial correctness of each P_i assuming that any calls made by P_i to any other procedure P_j (where possibly $i = j$) are correct. Then each P_i is partially correct.*

Proof. Define the procedure call tree for the execution of a procedure P_i as follows. Initially, there is one node in the tree, the root node, labelled by P_i. The root node is also initially the current node. During the execution of the program, whenever a procedure call is made, say to procedure P_j, a new child labelled by P_j is added to the current node, and that child becomes the current node. When a procedure finishes, the parent of the current node becomes the current node. Note that the procedure call tree has a node for each procedure call made during the run of the program and that if a procedure is called more than once, it will label more than one node in the call tree.

Clearly, if a call to P_i terminates, then its call tree is finite. We prove by induction on the height of the procedure call tree that every such call must be correct. For the base case, consider a tree of height zero. This means that P_i does not call any other procedures. Thus, the assumption that all other procedures P_j are correct is not necessary to justify the correctness of P_i. It follows that such calls to P_i are correct. For the induction step, suppose that all calls to procedures P_i are correct if their call tree has height k or less. Consider a call to a procedure P_i with call tree height $k + 1$. By the induction hypothesis, the result of each procedure call made by P_i is correct. Since every procedure called by P_i is correct, it must be the case that P_i is correct. □

Presenting a proof of an algorithm of this complexity is a significant challenge, and so every effort has been made to simplify the presentation. The approach is to identify the properties that are true between lines of code and then show that for every line of code, the preconditions of that line together with the execution of the line imply the postconditions.

We first define the notation used in the proof and give a list of properties used. Following that is a copy of the code annotated with the appropriate properties before

and after each line. Then, a detailed proof is presented for each line. Finally, we show how the postconditions of the main procedure, `AddFact`, guarantee that a decision procedure which uses the framework is sound and complete.

A.2 Definitions and Notation

A.2.1 The Shostak Theory

Some definitions and properties are only required (and only make sense) if a Shostak theory is included. For the purposes of this proof, we assume that a single Shostak theory $\mathcal{T}_\chi$ with signature Σ_χ is included with solver *solve* and canonizer *canon*. We will indicate which parts of the proof are dependent on this assumption (and could therefore be omitted if a Shostak theory is not included). We also assume that if multiple Shostak theories are to be included, they are first combined into a single Shostak theory. As discussed in Section 2.1.4, this is not always possible, but it often is for theories of interest. For convenience, we repeat the definitions of *solve* and *canon* from Section 3.4.3 here:

1. *canon* is a computable function from Σ_χ-terms to Σ_χ-terms, such that
   (a) $\mathcal{T}_\chi \models a = b$ iff $canon(a) \equiv canon(b)$
   (b) $canon(canon(t)) \equiv canon(t)$ for all terms t.
   (c) $canon(t)$ contains only variables occurring in t.
   (d) $canon(t) \equiv t$ if t is a variable.
   (e) If $canon(t)$ is a compound term, then $canon(c) \equiv c$ for each child c of $canon(t)$.

2. *solve* is a computable function from Σ_χ-equations to sets of Σ_χ-formulas defined as follows:
   (a) If $\mathcal{T}_\chi \models x \neq y$ then $solve(x = y) \equiv \{false\}$.
   (b) If $\mathcal{T}_\chi \models x = y$ then $solve(x = y) \equiv \emptyset$.

(c) Otherwise, $solve(x = y)$ returns a set of equations $\mathcal{E}$ in solved form such that $\mathcal{T} \models [(a = b) \leftrightarrow \exists \overline{w}.\mathcal{E}]$, where $\overline{w}$ is the set of variables which appear in $\mathcal{E}$ but not in a or b. Each of these variables must be fresh. We also require that for each $s = t \in \mathcal{E}$, $canon(t) \equiv t$.

As in Section 3.4.3, we define $canon_\chi(\alpha)$ as $\gamma_\chi^{-1}(canon(\gamma_\chi(\alpha)))$ and $solve_\chi(\beta)$ as $\gamma_\chi^{-1}(solve(\gamma_\chi(\beta)))$.

A.2.2 Nelson-Oppen Theories

We assume an arbitrary number of Nelson-Oppen style theories are included. Each Nelson-Oppen theory $\mathcal{T}_i$ provides a satisfiability procedure Sat_i satisfying the following specification for an arbitrary set of literals Θ:

$$Sat_i(\Theta) = \texttt{TRUE} \textit{ iff } \mathcal{T}_i \cup \gamma_i(\Theta) \not\models \textit{false}.$$

A.2.3 Variable Name Conventions

In the discussion and formulas that follow, we will consistently use the following variable conventions. Terms are represented by r, s, and t. The variables c, d, and e, are used to represent expressions (which could be formulas or terms). Theory indices are represented by i and j, and k and l are used as expression children indices. The variables W, X, Y, and Z are used for sets of formulas. Z is also used for sets of terms. Finally, sets of variables are often represented as a lowercase letter with a line over it, such as $\overline{w}$.

A.2.4 Program State

The state of the program is characterized by a number of variables which are abstractions of the program data structures and the program history. These are listed below.

1. The *assertion queue* $\mathcal{Q}$ is a set of formulas which are waiting to be asserted. Each call to `AddFact` initializes $\mathcal{Q}$ to a set containing the formula given as an

argument to `AddFact`. Theory-specific code may then add additional formulas to $\mathcal{Q}$. The formulas in $\mathcal{Q}$ are removed by one by `AddFact` and passed to `Assert`. Initially, $\mathcal{Q}$ is assumed to be empty.

2. As described in Section 3.5.1, the *non-literal set* $\mathcal{N}$ is a set of formulas which are not literals. It is initially empty.

3. The *inconsistent flag* $\mathcal{I}$ is a variable which indicates whether an inconsistency has been detected. It is initially `FALSE`.

4. The *find* and *notify* attributes for each expression are part of the program state. Initially, the *find* attribute of every expression is $\perp$, and the *notify* attribute of every expression is $\emptyset$.

5. The *assumption history*, $\mathcal{H}$, is the set of all formulas which have appeared as an argument to `AddFact`.

6. The *assertion database*, $\mathcal{A}$ is defined as the set of all formulas which have appeared as an argument to `TheoryAssert`. $\mathcal{A}_i$ denotes the subset consisting of the i-formulas in $\mathcal{A}$. Notice that $\mathcal{A}_i$ is exactly the set of formulas which appear as arguments to $\texttt{TheoryAssert}_i$.

7. As mentioned in Section 3.5.2, The code associated with an individual theory may introduce new formulas locally. To model this, we define for each theory a *theory-specific database* $\mathcal{B}_i$ which is an optional set of formulas maintained by the theory-specific code for theory i, with $\mathcal{B} = \bigcup \mathcal{B}_i$. We assume that initially $\mathcal{B}_i$ is empty for each i.

8. The *fact database* Φ is composed of the assertion database together with the theory-specific databases. Formally, $\Phi_i = \mathcal{A}_i \cup \mathcal{B}_i$, and $\Phi = \mathcal{A} \cup \mathcal{B} = \bigcup \Phi_i$.

9. We define the set of *shared terms* Λ as $\bigcup \Lambda_i$, where Λ_i is the set of all expressions which have appeared as arguments to $\texttt{TheoryAddSharedTerm}_i$.

10. The *Shostak database* is a set $\mathcal{S}$ of equations defined as follows. Initially, $\mathcal{S}$ is empty. Then, every time line 77 of `AssertEqualities` (according to the line-numbering of Section A.4) is executed (when $e[1]$ is not a compound χ-term), $\mathcal{S}$ is updated to be $\{e\}(\mathcal{S}) \cup \{e\}$. Note that $\{e\}(\mathcal{S})$ refers to the result of applying e as a substitution to $\mathcal{S}$ as described in Section 2.1.1. This piece of the program state is only required if a Shostak theory is included.

11. The *variable set* $\mathcal{V}$ is a set of variables which includes all the variables currently appearing in any program state. By representing this set explicitly, it is easy to express the fact that a fresh variable is different from any variable already in use.

In addition to these *global* state variables, there may be *local* state variables within a procedure. Local state variables include the arguments to the procedure and may include additional helper variables introduced within the procedure. The values of global and local state variables, together with the location of the program within the code form a sufficient description of the state of the program for the proof which follows. It is often necessary to compare the current state to the state at the beginning of a procedure. The current state will always be denoted just by the state variable itself. The state at the beginning of the procedure will be denoted by a primed version of the variable. Thus, Φ is the current state of the fact database, and Φ' refers to the state of the fact database at the beginning of the current procedure.

A.2.5 Other Definitions

The following definitions are either independent of the program state or derived from it. For convenience, we also repeat some definitions found in earlier chapters.

1. $\mathcal{T}_1, \ldots, \mathcal{T}_N$ are N stably infinite first-order theories, with signatures $\Sigma_1, \ldots, \Sigma_N$. $\mathcal{T} = \bigcup \mathcal{T}_i$ and $\Sigma = \bigcup \Sigma_i$.

2. $d \equiv e$ indicates that two expressions are syntactically identical. To express logical equivalence, we follow conventional notation, writing $d = e$ if d and e are

terms, and $d \leftrightarrow e$ if d and e are formulas. For convenience, we also introduce the notation $d \simeq e$ to indicate logical equivalence of arbitrary expressions. In other words, $d \simeq e$ means $d \leftrightarrow e$ if d and e are formulas, and $d = e$ if d and e are terms.

3. A *path* from an expression d to a sub-expression e of d is a sequence of expressions $e_0, e_1, \ldots, e_n$ such that $e_0 \equiv d$, e_{i+1} is a child of e_i for each $i, 0 \leq i < n$, and e is a child of e_n (in the degenerate case when $d \equiv e$, the path from d to e is the empty sequence).

4. $d \trianglelefteq e$ denotes that d is a sub-expression of e. Similarly, $d \triangleleft e$ denotes that d is a proper sub-expression of e.

5. Members of Σ_i are called i-symbols. A variable is called an i-variable if it is associated with $\mathcal{T}_i$. A Σ-term t is an i-term if it is an i-variable, a constant i-symbol, or an application of a functional i-symbol. An i-predicate is an application of a predicate i-symbol. An atomic i-formula is an an i-predicate or an equality whose left term is an i-term. An i-literal is an atomic i-formula or the negation of an atomic i-formula. An occurrence of a j-term t in either a term or a literal is i-alien if $i \neq j$ and all super-terms (if any) of that occurrence of t are i-terms. An i-term or i-literal is *pure* if the only non-logical symbols it contains are i-symbols and variables (i.e. only variables occur as i-alien sub-terms).

6. A j-term t occurs as an i-leaf in an expression e if every super-term of that occurrence (not including t) is an i-term and t is a variable or $i \neq j$. Note that a term t occurs as an i-alien of an expression e *iff* t is not an i-variable and occurs as an i-leaf of e. We say that a term t is an i-leaf if it occurs as an i-leaf in itself (i.e. it is a variable or a j-term, where $i \neq j$).

7. $\delta_i(e)$ is the set $\{c \mid c \text{ occurs as an } i\text{-leaf in } e\}$.

8. v is a mapping from Σ-terms to variables such that each i-term t is mapped to an i-variable $v(t)$ chosen from a set of variables not used anywhere else (i.e.

always disjoint from $\mathcal{V}$). $\gamma_i(\alpha)$ is the result of replacing all i-alien occurrences of terms t by $v(t)$. γ_i^{-1} denotes the inverse operation.

9. Given an equivalence relation $\sim$ with domain $dom_\sim$, we define $E_\sim = \{s = t \mid s,t \in dom_\sim \text{ and } s \sim t\}$, $D_\sim = \{s \neq t \mid s,t \in dom_\sim \text{ and } s \not\sim t\}$, and $Ar_\sim = E_\sim \cup D_\sim$.

10. We define the predicate *convex* to be true if and only if $\mathcal{T} \cup \Phi \models \mathcal{N}$ and for each atomic formula ϕ which is a sub-expression of some formula in $\mathcal{N}$, either $\phi \in \Phi$ or $\neg\phi \in \Phi$.

11. The *left-hand side* function *lhs* returns the set of all left-hand sides of a set of equations: $lhs(\mathcal{E}) = \{t \mid \exists\, e \in \mathcal{E}.\, t \equiv e[1]\}$.

12. Some procedures return an expression. In order to express properties which should hold for the returned expression, we use *retval* to denote the expression returned.

13. For an expression e, we define *hf*(e) (short for "has find") as follows: *hf*(e) iff $e.find \not\equiv \perp$.

14. The set *HF* is defined as $HF = \{e \mid hf(e)\}$.

15. The partial function $find^*$ is defined to be the expression obtained by following the find attributes of e until an expression is obtained whose find attribute is itself. If no such expression is obtained, then $find^*$ is undefined. A global invariant requires that $find^*$ be defined for all expressions in *HF* (see G_5, defined in Section A.3.1 below).

16. The relation $\sim$ is defined as follows. For expressions d and e, $d \sim e$ if and only if $hf(d) \wedge hf(e) \wedge find^*(d) \equiv find^*(e)$.

17. The relation $\sim_i$ is the restriction of $\sim$ to elements of Λ_i.

18. A sub-expression e of an expression d is called a *highest find-initialized* sub-expression of d if *hf*(e) and there exists a path from d to e such that for each expression e^* on the path, $\neg hf(e^*)$.

19. Let *fr* (short for "find-reduced") be a predicate on expressions defined as follows: $fr(d)$ iff $\mathit{find}^*(e) \equiv e$ for each highest find-initialized sub-expression e of d.

20. The *find database*, $\mathcal{F}$, is a set of equalities derived from the relation $\sim$ induced by the *find* attribute. It is defined as follows: $s = t \in \mathcal{F}$ iff $s \sim t$.

21. The set of *find-modified* terms $\mathcal{M}$ is $\{t \mid t.\mathit{find} \not\equiv t.\mathit{find}'\}$.

22. If a Shostak theory is included, then the set of *normal terms*, $\mathcal{R}$ is defined to be $\{t \mid \mathit{hf}(t) \wedge \mathit{find}^*(t) \equiv \mathit{canon}_\chi(\mathcal{S}(t))\}$.

As with the state variables, it is sometimes convenient to refer to the value of derived variables at the beginning of a procedure. Again, this is done by using a primed version of the variable.

A.3 Properties

A.3.1 Global Properties

Global properties are properties which are required to hold at every point in the code. Properties G_1 through G_{11} are required regardless of what theory-specific code is in use. Properties G_{12} through G_{20} are only required if theory-specific code for a Shostak theory is included. Since these global properties capture much of the intuition about how the algorithm works, a few words of explanation are included for each of them.

G_1. $\mathcal{I} \rightarrow (\mathcal{T} \cup \mathcal{H} \models \mathit{false})$.

This is the main soundness property. It states that the inconsistent flag is only set when the assumption history is in fact inconsistent.

G_2. $\mathcal{T} \cup \Phi \models \mathcal{F}$.

This property expresses soundness of the find database. The set of facts represented in the find database should be entailed by the fact database.

G_3. $\mathcal{T} \cup \mathcal{H} \models \exists \overline{w}.\, \Phi$, where $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\mathcal{H})$.

The framework allows fresh variables to be introduced at various places, so the

fact database is not logically equivalent to the assumption history, but it is required to be equisatisfiable as expressed by this property.

G_4. $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \mathcal{Q} \cup \mathcal{N}$.

This property expresses soundness of the assertion queue and the non-literal set. They should be entailed by the assumption history together with the fact database.

G_5. $\forall e.\, \mathit{hf}(e) \rightarrow \mathit{find}^*(e)$ is defined.

This property requires the *find* data structure to be well-defined. Chains of *find* attributes should always be defined and end at an expression whose *find* attribute is itself.

G_6. $\forall e.\, \mathit{hf}(e) \rightarrow \mathit{hf}(c)$ for each child c of e.

This property requires that the children of expressions with *find* pointers always have find pointers as well.

G_7. $\forall e.\, \mathit{hf}(e) \rightarrow e$ is a term.

Only terms are allowed to have find pointers.

G_8. $\mathit{free}(\mathcal{Q} \cup \mathcal{N} \cup \Phi \cup \mathcal{H} \cup \mathcal{F} \cup \Lambda) \subseteq \mathcal{V}$.

The set $\mathcal{V}$ should include all free variables appearing in the program state.

G_9. $\forall t, j.$ if t occurs j-alien term in any sub-term of $\Phi \cup \mathcal{F}$, then $(t \in \Lambda_j \wedge t \in \Lambda_{\mathcal{T}(t)})$.

Every theory knows about any of its own terms that are shared as well as any shared terms appearing in any of its terms or literals. Notice that this property also implies that all terms which occur as aliens are contained in Λ.

G_{10}. If G_{10}^{ok} then $\forall t, j.$ if $t \in \Lambda_j$, where $j \neq \mathcal{T}(t)$, then t occurs j-alien in some sub-term of $\mathcal{A}$.

This is essentially the converse of part of the previous property. If t is identified as a shared term by theory $\mathcal{T}_j$, then t had better really occur as a j-alien term somewhere in the assertion database. Unfortunately, this property is not quite global. To compensate for this, we introduce a state variable, G_{10}^{ok} which is

defined to be true everywhere except at lines 101 through 107, lines 114 through 146, and inside any call to `TheoryAddSharedTerm` or `TheorySetup`.

G_{11}. $\forall\, t \lhd \mathcal{A}.\, hf(t)$.
All terms in the assertion database have find pointers.

G_{12}. $\mathcal{B}_\chi = \emptyset$.
The Shostak theory does not maintain any theory-specific formulas.

G_{13}. $\mathcal{S}$ is in χ-solved form.
The Shostak database is in solved form.

G_{14}. $\mathcal{T} \cup \mathcal{F} \models \mathcal{S}$.
The Shostak database is entailed by the find database.

G_{15}. $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(\mathcal{F})$.
The (Shostak-pure) find database is entailed by the (Shostak-pure) Shostak database.

G_{16}. If G_{16}^{ok} then
$\forall\, t.\, [(hf(t) \wedge t \text{ is a compound } \chi\text{-term}) \rightarrow (\forall\, c \in \delta_\chi(t).\, (\chi, t) \in c.notify)]$.
Each compound term in the Shostak theory that has a find pointer is required to be on the notify list of all terms which occcur as a χ-leaf in the term. This ensures that if any of these leaves is asserted equal to something else, the Shostak `TheoryUpdate` method will be called. Again, this property is not quite global, so we introduce the state variable G_{16}^{ok} which is defined to be true everywhere except at line 142 and inside any call to `TheorySetup`.

G_{17}. $\forall\, t, c.\, [(\chi, t) \in c.notify) \rightarrow (t \text{ is a compound } \chi\text{-term} \wedge c \in \delta_\chi(t) \wedge hf(t)]$.
This property expresses essentially the converse of the above property, limiting the Shostak data on the notify list to only the cases covered in the above property.

G_{18}. $\forall\, e.\, [(\forall\, c \in \delta_\chi(e).\, fr(c)) \rightarrow fr(e)]$.
A non-obvious invariant for Shostak terms is that if all terms which occur as

leaves in them are find-reduced, then they are also find-reduced. This invariant makes it easy to guarantee that Shostak terms are find-reduced.

G_{19}. $\forall t.$ if t is a χ-leaf, then
$[(hf(t) \rightarrow (t.find \not\equiv t \leftrightarrow t \in lhs(\mathcal{S}))$ and $(t \lhd \mathcal{S} \rightarrow hf(t))]$.
Each term with a find pointer which is foreign to the Shostak theory has the property that it points to itself if and only if it does not appear on the left-hand side of an equation in $\mathcal{S}$ (i.e. it is not a solitary variable of $\gamma_\chi(\mathcal{S})$). Also, all terms in $\mathcal{S}$ have find pointers.

G_{20}. $\forall e \in \mathcal{A}.e$ is a literal.
The assertion database $\mathcal{A}$ consists only of literals.

A.3.2 Preservation Properties

$P_=(X, Y, \ldots)$. This is a special property which expresses that each state variable in its parameter list currently has the same value that it had at the beginning of the procedure: $X' = X \wedge Y' = Y \ldots$.

$P_\subseteq(X, Y, \ldots)$. This property is similar to the above property, except that it expresses that each state variable in its parameter list is a superset of what it was at the beginning of the procedure: $X' \subseteq X \wedge Y' \subseteq Y \ldots$.

A.3.3 Other Properties

We now list the other properties which will be used to annotate the code. They express conditions on the global and local state which are required to hold at specific points in the program.

Each item in the following list consists of a property abbreviation followed by the property definition. The property abbreviation is used to refer to the property in the code which follows and consists of the letter "P" with a numeric subscript followed by a list of the global state (if any) on which the property depends in square brackets. For convenience, we use the notation $*.find$ to refer to all find pointers, $*.notify$ to

refer to all notify lists, and Λ_* to refer to Λ_i for all i. We also use *all* to denote all of the global state variables.

Most properties also depend on local state variables. To enable reuse of such properties, the property list below is parameterized. The variables shown in parentheses next to the property abbreviation are parameters to the properties. Thus, when the properties are used in context, the actual local state variables appearing as arguments to the properties should be substituted for the parameters.

$P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$. $\neg\mathcal{I} \rightarrow (\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H})$.

$P_2(e)$. e is a Σ-formula.

$P_3[\Phi, \mathcal{H}](e)$. $\mathit{free}(e) \cap \mathit{free}(\Phi - \mathcal{H}) = \emptyset$.

$P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e)$. $\neg\mathcal{I} \rightarrow (\mathcal{T} \cup \mathcal{N} \cup \Phi \cup \{e\} \models \mathcal{H})$.

$P_5[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{Q}](e)$. $(P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e) \wedge \mathcal{Q} = \{e\}) \vee (P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}] \wedge \mathcal{Q} \neq \emptyset)$.

$P_6[\Phi, \mathcal{H}](e)$. $\mathcal{T} \cup \mathcal{H} \cup \Phi \models e$.

$P_7[\Phi, \Lambda_*, *.\mathit{find}](i)$. $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ is satisfiable.

$P_8[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](i)$.
$(\mathcal{Q} = \emptyset \wedge \neg\mathcal{I} \wedge \mathit{convex}) \rightarrow (\forall\, j, 0 < j < i.\ P_7[\Phi, \Lambda_*, *.\mathit{find}](j))$.

$P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](i)$. $(\mathcal{Q} = \emptyset \wedge \neg\mathcal{I} \wedge \mathit{convex}) \rightarrow P_7[\Phi, \Lambda_*, *.\mathit{find}](i))$.

$P_{10}[\Phi, \mathcal{I}, \mathcal{N}, \Lambda_*, *.\mathit{find}]$. $(\neg\mathcal{I} \wedge \mathit{convex}) \rightarrow (\forall\, i, 0 < i \leq N.\ P_7[\Phi, \Lambda_*, *.\mathit{find}](i))$.

$P_{11}(e)$. e is a literal.

$P_{12}[\Phi, \mathcal{H}](e)$. $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \exists\, \overline{w}.\ e$, where $\overline{w} = \mathit{free}(e) - \mathit{free}(\mathcal{H} \cup \Phi)$.

$P_{13}(X)$. $\forall\, d \in X.[P_{11}(d) \wedge Op(d) = \text{`='} \wedge (\forall\, e \in X.d \not\equiv e \rightarrow (d[1] \not\equiv e[1] \wedge d[1] \not\equiv e[2]))]$.

$P_{14}[*.\mathit{find}](X)$. $\mathit{false} \in X \vee (\forall\, e \in X.\ \mathit{fr}(e) \wedge P_{13}(X))$.

$P_{15}[\Phi, \mathcal{N}](W, X, Y)$. $W \subseteq \Phi \wedge X \subseteq \mathcal{N} \wedge Y \subseteq \Phi$.

$P_{16}[*.\mathit{find}](X,Y)$.
$\forall\, e \in Y.\, [(e \in X \rightarrow (e[1].\mathit{find} \equiv e[1] \wedge e[2].\mathit{find} \equiv e[2])) \wedge (e \notin X \rightarrow \mathit{fr}(e))]$.

$P_{17}[\Phi](X)$. For $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(X)$, $\mathcal{T} \cup X \models \exists\, \overline{w}.\, \Phi$ and $\overline{w} \cap \mathcal{V}\,' = \emptyset$.

$P_{18}[*.\mathit{find}](X)$. $\forall\, e \in X.\, (e[1].\mathit{find} \equiv e[1] \wedge e[2].\mathit{find} \equiv e[2])$.

$P_{19}[*.\mathit{find}](X,Y)$.
$\forall\, e \in Y.\, [e[2].\mathit{find} \equiv e[2] \wedge (e \in X \rightarrow e[1].\mathit{find} \equiv e[2]) \wedge$
$(e \notin X \rightarrow e[1].\mathit{find} \equiv e[1])]$.

$P_{20}[\mathcal{H}](W^*, e)$.
$\exists\, W, X, Y, Z.$ $\{W, X, Z\}$ *is a partition of* $W^* \wedge$
$\mathcal{T} \cup \mathcal{H} \models \exists\, \overline{w}.\, W,$ *where* $\overline{w} = \mathit{free}(W) - \mathit{free}(\mathcal{H}) \wedge$
$\mathcal{T} \cup \mathcal{H} \cup W \models \exists\, \overline{x}, \overline{y}.\, (X \cup Y)$,
where $\overline{x} = \mathit{free}(X) - \mathit{free}(\mathcal{H} \cup W)$
and $\overline{y} = \mathit{free}(Y) - \mathit{free}(\mathcal{H} \cup W \cup X) \wedge$
$\mathcal{T} \cup W \cup X \models \exists\, \overline{z}.\, Z \ \wedge\ \overline{z} \cap \mathit{free}(\mathcal{H} \cup \{e\}) = \emptyset,$
where $\overline{z} = \mathit{free}(Z) - \mathit{free}(W \cup X) \wedge$
$e \in Y$.

$P_{21}[*.\mathit{find}]$. $\forall\, e.\, (e.\mathit{find} \equiv e \vee e.\mathit{find} \equiv e.\mathit{find}')$.

$P_{22}[*.\mathit{find}](Z)$. $\forall\, t \in Z.\, t.\mathit{find} \equiv t$.

$P_{23}(e)$. e is a term.

$P_{24}[\Lambda_*](Z,e)$.
$\forall\, t, d.\, [(t \trianglelefteq Z \wedge t \triangleleft d \trianglelefteq e \wedge t$ occurs $\mathcal{T}(d)$-alien in $d) \rightarrow (t \in \Lambda_{\mathcal{T}(t)} \wedge t \in \Lambda_{\mathcal{T}(d)})]$.

$P_{25}[*.\mathit{find}](e)$. $\forall\, t, t$ a maximal sub-term of $e.\, t.\mathit{find} \equiv t$.

$P_{26}[\Lambda_*](e,i)$. $\mathcal{T}(e) \neq i \rightarrow (e \in \Lambda_{\mathcal{T}(e)} \wedge e \in \Lambda_i)$.

$P_{27}[*.\mathit{find}](t)$. $\forall\, s.\, (s \not\trianglelefteq t \rightarrow s.\mathit{find} \equiv s.\mathit{find}')$.

$P_{28}[*.\mathit{find}](t)$. $\forall\, s.\, [s \trianglelefteq t \rightarrow (s.\mathit{find} \equiv s \vee s.\mathit{find} \equiv s.\mathit{find}')]$.

$P_{29}[*.\mathit{find}](e,k)$. $\forall\, l, 1 \leq l < k.\ e[l].\mathit{find} \equiv e[l]$.

$P_{30}[*.\mathit{find}](e,k)$. $\forall\, l, 1 \leq l < k.\ \mathit{fr}(e[l])$.

$P_{31}[*.\mathit{find}](e)$. $\forall\, k, 1 \leq k \leq \mathit{Arity}(e).\ e[k].\mathit{find} \equiv e[k]$.

$P_{32}[\Lambda_*](e,k)$. $\forall\, l, 1 \leq l < k.\ P_{24}[\Lambda_*](e[l], e[l]) \wedge P_{26}[\Lambda_*](e[l], \mathcal{T}(e))$.

$P_{33}(e)$. e is a term or e is an atomic formula.

$P_{34}[\mathit{all}](i)$. $P_{=}(\Phi - \mathcal{B}_i, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, \Lambda_*, *.\mathit{find}) \wedge P_{\subseteq}(\mathcal{Q}, \mathcal{B}_i, \mathcal{V}, *.\mathit{notify})$ and
$\mathcal{T}_i \cup \gamma_i(\Phi_i' \models \exists\, \overline{w}.\ \mathcal{B}_i) \wedge \overline{w} \cap \mathcal{V}\,' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$, where $\overline{w} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i')$ and
$\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}')) \wedge \mathit{free}(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}$.

$P_{35}[\Phi](d,e)$. For $\overline{w} = \mathit{free}(e) - \mathit{free}(d)$, $\mathcal{T} \cup \Phi \models d \leftrightarrow \exists\, \overline{w}.e$ and $\overline{w} \cap (\mathcal{V}\,' \cup \mathit{free}(\Phi)) = \emptyset$.

$P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i)$. $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i})$ is satisfiable.

$P_{37}[\Phi, \Lambda_*, *.\mathit{find}](i,e)$. $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i}) \models \gamma_i(e)$.

$P_{38}[\mathcal{A}, \Lambda_*](e)$. $\forall\, j. \forall\, t \in \Lambda_j.[j \neq \mathcal{T}(t) \rightarrow (t$ occurs j-alien in some sub-term of $\mathcal{A} \cup \{e\}]$.

$P_{39}[\Lambda_*](t)$. $\forall\, j. \forall\, s \not\trianglelefteq t.(s \in \Lambda_j \leftrightarrow s \in \Lambda'_j)$.

$P_{40}(s,t,i,j)$. $(s \in \Lambda'_j) \vee (s \equiv t \wedge j = i) \vee (s$ occurs j-alien in some sub-term of $t)$.

$P_{41}[\Lambda_*](s,t,i,j)$. $(s \in \Lambda_j \wedge j \neq \mathcal{T}(s)) \rightarrow P_{40}(s,t,i,j)$.

$P_{42}[\Lambda_*](t,i)$. $\forall\, s \trianglelefteq t. \forall\, j. P_{40}(s,t,i,j)$.

The following properties are only required if theory-specific code for a Shostak theory is included. To distinguish them from the previous properties, they use the letter "S" instead of "P".

$S_1[\mathcal{S}](s,t)$. $s \equiv \mathit{canon}_\chi(\mathcal{S}(t))$.

$S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$. $\neg\mathcal{I} \rightarrow [\forall\, e.\ (\mathit{hf}(e) \rightarrow S_1[\mathcal{S}](\mathit{find}^*(e), e))]$.

$S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$. $\neg\mathcal{I} \rightarrow [\forall\, e \in \mathcal{A}.\ (\mathit{Op}(e) = \text{`='} \rightarrow e[1] \sim e[2])]$.

$S_4[\mathcal{S}, *.find](e)$. $\forall\, t \trianglelefteq e.\ (\neg hf(t) \rightarrow S_1[\mathcal{S}](t,t))$.

$S_5(e)$. $Op(e) = \text{`='} \rightarrow e[1] \not\equiv e[2]$.

$S_6(X)$. X is in χ-solved form.

$S_7[\mathcal{S}, *.find](X)$. $false \in X \vee (S_4[\mathcal{S}, *.find](X) \wedge S_6(X))$.

$S_8(t)$. t is a compound χ-term.

$S_9[*.find](e)$. $\exists\, c \in \delta_\chi(e).\ (c.find \not\equiv c)$.

$S_{10}[\mathcal{S}, *.find](X)$. $\exists\, e.[X = \{e\} \wedge S_8(e[1]) \wedge hf(e[1]) \wedge S_1[\mathcal{S}](e[2], e[1]) \wedge S_9[*.find](e[1])]$.

$S_{11}[\mathcal{S}, *.find](X)$. $\forall\, e \in X.\ [S_1[\mathcal{S}](e[2], e[2]) \wedge fr(e) \wedge S_4[\mathcal{S}, *.find](e)]$.

$S_{12}[\mathcal{S}, *.find](X)$. $false \in X \vee ((S_6(X) \vee S_{10}[\mathcal{S}, *.find](X)) \wedge S_{11}[\mathcal{S}, *.find](X))$.

$S_{13}[*.find](X)$. $\forall\, e \in X.\ (Op(e) = \text{`='} \rightarrow e[1] \sim e[2])$.

$S_{14}[*.find](\mathcal{R}^*, X)$. $(\mathcal{M} - \mathcal{R}^*) \subseteq lhs(X)$.

$S_{15}[\mathcal{S}, *.find](X, Y)$. $S_{10}[\mathcal{S}, *.find](X) \vee [S_6(X) \wedge (\forall\, e \in (X - Y).\ \mathcal{S}(e) \equiv e) \wedge$
$(\forall\, e \in X.\ S_1[\mathcal{S}](e[2], e[2]))]$.

$S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, X, Y, \mathcal{S}^*)$.
$\forall\, t \in \mathcal{R}^*.\ [hf(t) \wedge$
$(\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(Y) = \emptyset \vee \neg S_6(X) \rightarrow$
$find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \equiv canon_\chi(\mathcal{S}(t))) \wedge$
$(\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(Y) \neq \emptyset \wedge \neg S_8(canon_\chi(\mathcal{S}^*(t))) \rightarrow$
$find^*(t) \equiv canon_\chi(\mathcal{S}(t)) \wedge \delta_\chi(canon_\chi(\mathcal{S}(t))) \cap lhs(X) = \emptyset) \wedge$
$(\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(Y) \neq \emptyset \wedge S_8(canon_\chi(\mathcal{S}^*(t))) \rightarrow$
$find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)))]$.

$S_{17}(X, Y)$. $(S_6(X) \rightarrow (Y \subseteq \mathcal{S})) \wedge (Y \subseteq X)$.

$S_{18}[\mathcal{S}, *.find](\mathcal{R}^*)$. $\mathcal{M} - \mathcal{R}^* \subseteq \mathcal{R}$.

$S_{19}[\mathcal{S}, *.find](\mathcal{R}^*, X, \mathcal{S}^*)$.
$\forall\, t \in \mathcal{R}^*.\, [t \in \mathcal{R} \vee (S_8(canon_\chi(\mathcal{S}^*(t))) \wedge hf(t) \wedge find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \wedge \delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) \neq \emptyset)]$.

$S_{20}[\mathcal{S}](X)$. $\forall\, e \in X.\, (e[1] \in lhs(\mathcal{S}) \vee S_8(e[1]))$.

$S_{21}[*.find](\mathcal{L}, c)$. $\forall\, t.\, [((\chi, t) \in \mathcal{L}) \rightarrow (hf(t) \wedge S_8(t) \wedge\ c \in \delta_\chi(t))]$.

$S_{22}[\mathcal{I}, \mathcal{S}, *.find](e, \mathcal{R}^*, X, \mathcal{U}, \mathcal{S}^*)$. $\mathcal{I} \vee$
$[\forall\, t \in \mathcal{R}^*.\, t \in \mathcal{R} \vee (S_8(canon_\chi(\mathcal{S}^*(t))) \wedge hf(t) \wedge find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \wedge \delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) \neq \emptyset \wedge e[1] \in \delta_\chi(canon_\chi(\mathcal{S}^*(t))) \rightarrow (\chi, canon_\chi(\mathcal{S}^*(t))) \in \mathcal{U})]$.

$S_{23}[*.find, *.notify](e)$.
$\forall\, t \not\equiv e.\, [(hf(t) \wedge S_8(t)) \rightarrow (\forall\, c \in \delta_\chi(t).\, (\chi, t) \in c.notify)]$.

$S_{24}[\mathcal{S}, *.find](e, k)$. $\forall\, l, 1 \leq l < k.\, S_4[\mathcal{S}, *.find](e[l])$.

$S_{25}[\mathcal{S}, *.find](e)$. $\forall\, l, 1 \leq l \leq Arity(e).\, S_4[\mathcal{S}, *.find](e[l])$.

$S_{26}[\mathcal{S}, *.find](d, e)$.
$(d \equiv e \rightarrow S_4[\mathcal{S}, *.find](e)) \wedge (d \not\equiv e \rightarrow S_{25}[\mathcal{S}, *.find](d))$.

$S_{27}(d, e)$. $d \not\equiv e \vee S_5(e)$.

$S_{28}[*.find](i, e)$. $i = \chi \rightarrow (hf(e) \wedge S_8(e) \wedge S_9[*.find](e))$.

$S_{29}[\mathcal{I}, \mathcal{S}](X, i)$. $\mathcal{I} \vee [\forall\, e \in X.\, (Op(e) = \text{'}\neg\text{'} \rightarrow (\mathcal{T}_i \cup \gamma_i(\mathcal{S}) \not\models \gamma_i(e[1])))]$.

$S_{30}[*.find](e)$. $\forall\, c \in \delta_\chi(e).fr(c)$.

$S_{31}[\mathcal{S}, *.find](e)$. $\forall\, c \in \delta_\chi(e).S_4[\mathcal{S}, *.find](c)$.

$S_{32}[\mathcal{S}](e)$. $\forall\, c \in \delta_\chi(e).\mathcal{S}(c) \equiv c$.

$S_{33}[\mathcal{S}](d, e)$. $canon_\chi(d) \equiv canon_\chi(\mathcal{S}(e))$.

$S_{34}[\mathcal{I}, \mathcal{S}, *.find](d)$. $\mathcal{I} \vee (d.find \not\equiv d) \vee (d \in \mathcal{R})$.

$S_{35}[*.notify](e)$. $S_8(e) \rightarrow \forall\, c \in \delta_\chi(e).(\chi, e) \in c.notify$.

$S_{36}[*.notify](Z)$. $\forall\, c \in Z.(\chi, e) \in c.notify$.

$S_{37}[*.find](e)$. $\forall\, k, 1 \leq k \leq Arity(e).\ hf(e[k]) \vee fr(e[k])$.

$S_{38}[*.find](e, k)$. $\forall\, l, 1 \leq l < k.\ S_{30}[*.find](e[l])$.

$S_{39}[\mathcal{S}, *.find](e, k)$. $\forall\, l, 1 \leq l < k.\ S_{31}[\mathcal{S}, *.find](e[l]) \wedge S_{32}[\mathcal{S}](e[l])$.

$S_{40}[\mathcal{S}, *.find](e, k)$. $\forall\, l, k \leq l \leq Arity(e).\ (S_4[\mathcal{S}, *.find](e[l]) \wedge e'[l] \equiv e[l])$.

$S_{41}[\mathcal{S}](e, k)$. $\forall\, l, 1 \leq l < k.\ S_{33}[\mathcal{S}](e[l], e'[l])$.

$S_{42}[\mathcal{S}](d, e)$. If e is a term then $d \equiv canon_\chi(\mathcal{S}(e))$.

A.4 Annotated Code

The code shown below includes the framework code with non-convex extensions, and the theory-specific code for Nelson-Oppen and Shostak theories, including the default implementations (see Figure 3.6) of the procedures not explicitly provided (note that the modification of global state variables by `TheoryAddSharedTerm`$_i$ and `TheoryAssert`$_i$ has been moved to the framework code—see the note on "virtual" code below). The actual code shown below is equivalent in function to that given in Chapter 3, but in some cases it is slightly modified to be more explicit or give more detail.

Before and after each line of code is a list of properties which should hold at that point in the execution of the program. Since the global properties always hold, a single capital "G" at each line is used to represent all of them.

Also, it is often helpful to add "virtual" lines of code which describe how some piece of local or global state is modified by executing a particular line of code. These virtual pieces of code are shown in square brackets. To aid readability, everything that is not an actual part of the code is shown in half-tone gray. This makes it easier to separate the code from the annotations.

Following the presentation of the code, we give proofs for the properties between each line of code. However, since many of the proofs are trivial, only those properties which do not follow trivially are proved in detail. These properties are underlined in the annotated code to make it easy to see at a glance which properties require proof. Note that all global properties depend only on global states variables, so they are always trivially preserved unless some global state is modified. Also, when verifying the properties following a procedure call, we also verify that the preconditions of the procedure are satisfied. Because the proofs of properties following a procedure call can often be subtle, we always justify each of them (and thus, all properties following procedure calls are always underlined).

A.4.1 Framework Code

0. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_2(e),\ P_3[\Phi,\mathcal{H}](e),\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
1. `AddFact(`e`)` [$\mathcal{H}$:= $\mathcal{H} \cup \{e\}$; $\mathcal{V}$:= $\mathcal{V} \cup$ *free*(e);]
2. $\underline{G},\ \underline{P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)},\ e \in \mathcal{H},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
3. $\mathcal{Q}$:= $\{e\}$;
4. $\underline{G},\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e),\ \mathcal{Q} = \{e\},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
5. REPEAT
6. $G,\ \underline{P_5[\Phi,\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{Q}](e)},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
7. WHILE $\mathcal{Q} \neq \emptyset$ AND $\neg\mathcal{I}$ DO BEGIN
8. $G,\ \mathcal{Q} \neq \emptyset,\ \neg\mathcal{I},\ \underline{P_5[\Phi,\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{Q}](e)},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
9. Choose $e^* \in \mathcal{Q}$;
10. $G,\ \underline{P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)},\ \mathcal{Q} \neq \emptyset,\ \neg\mathcal{I},\ e^* \in \mathcal{Q},$
$\underline{P_6[\Phi,\mathcal{H}](e^*)},\ \underline{\mathit{free}(e^*) \subseteq \mathcal{V}},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
11. $\mathcal{Q}$:= $\mathcal{Q} - \{e^*\}$;
12. $\underline{G},\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*),\ \neg\mathcal{I},\ P_6[\Phi,\mathcal{H}](e^*),\ \mathit{free}(e^*) \subseteq \mathcal{V},$
$S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
13. Assert(e^*);
14. $\underline{G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]}$
15. END
16. $G,\ \underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
17. FOR $\underline{i := 1 \text{ TO } N}$ DO BEGIN
18. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](i)},\ S_2[\mathcal{I},\mathcal{S},*.find],$
$S_3[\mathcal{A},\mathcal{I},*.find]$
19. IF $\mathcal{Q} = \emptyset$ AND $\neg\mathcal{I}$ AND *convex* THEN BEGIN
20. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \mathcal{Q} = \emptyset,\ \neg\mathcal{I},\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](i),\ \mathit{convex},$
$S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
21. TheoryCheckSat$_i$();
22. $\underline{G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](i),}$
$\underline{P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](i),\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]}$
23. END
24. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](i),$
$P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](i),\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
25. END
26. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](N+1)},$
$P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](i),\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
27. UNTIL $\mathcal{Q} = \emptyset$ OR $\mathcal{I}$;
28. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{P_{10}[\Phi,\mathcal{I},\mathcal{N},\Lambda_*,*.find]},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
29. END AddFact
30. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_{10}[\Phi,\mathcal{I},\mathcal{N},\Lambda_*,*.find],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$

31. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e)$, $free(e) \subseteq \mathcal{V}$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
32. `Assert(`e`)`
33. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e)$, $free(e) \subseteq \mathcal{V}$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
34. e^* `:= Simplify(`e`);`
35. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e^*)$, $free(e^*) \subseteq \mathcal{V}$, $\mathcal{T} \cup \Phi \models e \simeq e^*$, $fr(e^*)$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$, $S_4[\mathcal{S},*.find](e^*)$, $S_5(e^*)$
36. `IF` e^* `is not a literal THEN BEGIN`
37. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e^*)$, $free(e^*) \subseteq \mathcal{V}$, $fr(e^*)$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
38. $\mathcal{N}$ `:=` $\mathcal{N} \cup \{e^*\}$`;`
39. G, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
40. `END ELSE IF` $Op(e^*) =$ `'='` `THEN BEGIN`
41. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e^*)$, $free(e^*) \subseteq \mathcal{V}$, $fr(e^*)$, $P_{11}(e^*)$, $Op(e^*) =$ `'='`, $e^*[1] \not\equiv e^*[2]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$, $S_4[\mathcal{S},*.find](e^*)$
42. $\mathcal{E}$ `:= TheorySolve(`e^*`);`
43. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E})$, $\neg\mathcal{I}$, $free(\mathcal{E}) \subseteq \mathcal{V}$, $P_{12}[\Phi,\mathcal{H}](\mathcal{E})$, $P_{14}[*.find](\mathcal{E})$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$, $S_7[\mathcal{S},*.find](\mathcal{E})$
44. `AssertEqualities(`$\mathcal{E}$`);`
45. G, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
46. `END ELSE IF` $e^* \equiv$ *false* `THEN BEGIN`
47. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e^*)$, $\mathcal{T} \cup \Phi \models e \simeq e^*$, $fr(e^*)$, $e^* \equiv false$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
48. $\mathcal{I}$ `:= TRUE;`
49. G, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
50. `END ELSE IF` $e^* \not\equiv$ *true* `THEN BEGIN`
51. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e^*)$, $free(e^*) \subseteq \mathcal{V}$, $fr(e^*)$, $P_{11}(e^*)$, $Op(e^*) \neq$ `'='`, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$, $S_4[\mathcal{S},*.find](e^*)$
52. `AssertFormula(`e^*`);`
53. G, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
54. `END`
55. G, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$
56. `END Assert`
57. G, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$

58. $G,\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E}),\ \neg\mathcal{I},\ P_{12}[\Phi,\mathcal{H}](\mathcal{E}),\ P_{14}[*.\mathit{find}](\mathcal{E}),\ \mathit{free}(\mathcal{E})\subseteq\mathcal{V},$
$S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ S_{12}[\mathcal{S},*.\mathit{find}](\mathcal{E})$
59. `AssertEqualities(`$\mathcal{E}$`)`
60. $G,\ P_=(\mathit{all}),\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E}),\ \neg\mathcal{I},\ \mathit{free}(\mathcal{E})\subseteq\mathcal{V},\ P_{12}[\Phi,\mathcal{H}](\mathcal{E}),\ P_{14}[*.\mathit{find}](\mathcal{E}),$
$S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ S_{12}[\mathcal{S},*.\mathit{find}](\mathcal{E})$
61. `IF` $\mathit{false}\in\mathcal{E}$ `THEN BEGIN`
62. $G,\ P_=(\mathit{all}),\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E}),\ P_{12}[\Phi,\mathcal{H}](\mathcal{E}),\ \mathit{false}\in\mathcal{E},\ P_{14}[*.\mathit{find}](\mathcal{E})$
63. $\mathcal{I}$ `:= TRUE;`
64. $\underline{G},\ P_=(\mathit{all}-\{\mathcal{I}\}),\ \underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]},\ \mathcal{I}$
65. `END ELSE BEGIN` [$\mathcal{A}^*$ `:=` $\mathcal{A}$; Φ^* `:=` Φ; $\mathcal{N}^*$ `:=` $\mathcal{N}$; X `:=` $\emptyset$]
66. $G,\ P_=(\mathit{all}),\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E}),\ \neg\mathcal{I},\ \mathit{free}(\mathcal{E})\subseteq\mathcal{V},\ P_{12}[\Phi,\mathcal{H}](\mathcal{E}),\ \mathit{false}\notin\mathcal{E},$
$\underline{\forall e\in\mathcal{E}.\,\mathit{fr}(e)},\ \underline{P_{13}(\mathcal{E})},\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ \underline{S_6(\mathcal{E})\vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E})},$
$\underline{S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E})}$
67. `FOREACH` $e\in\mathcal{E}$ `DO BEGIN`
68. $G,\ P_=(\mathcal{H},\mathcal{S}),\ P_\subseteq(\mathcal{V},\mathcal{F},\mathcal{R}),\ \underline{P_{15}[\Phi,\mathcal{N}](\Phi^*,\mathcal{N}^*,X)},\ P_{13}(\mathcal{E}),\ X\subseteq\mathcal{E},$
$\underline{P_{16}[*.\mathit{find}](X,\mathcal{E})},\ \underline{P_{17}[\Phi](\Phi^*\cup X)},\ \mathit{free}(\mathcal{E})\subseteq\mathcal{V}',\ S_6(\mathcal{E})\vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E}),$
$S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E}),\ \underline{\mathcal{A}=\mathcal{A}^*\cup X},\ \underline{S_{13}[*.\mathit{find}](\mathcal{A}^*)},\ \underline{\mathcal{M}\subseteq\mathcal{R}},\ e\in\mathcal{E},\ e\notin X$
69. `AssertFormula(`e`);` [X `:=` $X\ \cup\ \{e\}$;]
70. $\underline{G,\ P_=(\mathcal{H},\mathcal{S}),\ P_\subseteq(\mathcal{V},\mathcal{F},\mathcal{R}),\ P_{15}[\Phi,\mathcal{N}](\Phi^*,\mathcal{N}^*,X),\ P_{13}(\mathcal{E}),\ X\subseteq\mathcal{E},}$
$\underline{P_{16}[*.\mathit{find}](X,\mathcal{E}),\ P_{17}[\Phi](\Phi^*\cup X),\ \mathit{free}(\mathcal{E})\subseteq\mathcal{V}',\ S_6(\mathcal{E})\vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E}),}$
$\underline{S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E}),\ \mathcal{A}=\mathcal{A}^*\cup X,\ S_{13}[*.\mathit{find}](\mathcal{A}^*),\ \mathcal{M}\subseteq\mathcal{R}}$
71. `END`
72. $G,\ P_=(\mathcal{S}),\ P_\subseteq(\mathcal{F},\mathcal{R}),\ \underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]},\ P_{13}(\mathcal{E}),\ \underline{P_{18}[*.\mathit{find}](\mathcal{E})},\ \mathcal{A}=\mathcal{A}^*\cup\mathcal{E},$
$S_6(\mathcal{E})\vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E}),\ \underline{S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E})},\ \underline{S_{13}[*.\mathit{find}](\mathcal{A}^*)},\ \mathcal{M}\subseteq\mathcal{R},\ \underline{\mathcal{E}\subseteq\Phi}$
73. [X `:=` $\emptyset$; $\mathcal{R}^*$ `:=` $\mathcal{R}$; $\mathcal{S}^*$ `:=` $\mathcal{S}$]
74. $G,\ P_=(\mathcal{S}),\ P_\subseteq(\mathcal{F},\mathcal{R}),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_{13}(\mathcal{E}),\ P_{18}[*.\mathit{find}](\mathcal{E}),\ \mathcal{A}=\mathcal{A}^*\cup\mathcal{E},$
$S_6(\mathcal{E})\vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E}),\ S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E}),\ S_{13}[*.\mathit{find}](\mathcal{A}^*),\ \mathcal{M}\subseteq\mathcal{R},\ \mathcal{E}\subseteq\Phi$
75. `FOREACH` $e\in\mathcal{E}$ `DO BEGIN`
76. $G,\ P_\subseteq(\mathcal{F},\mathit{lhs}(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_{13}(\mathcal{E}),\ \mathcal{E}\subseteq\Phi,\ \underline{P_{19}[*.\mathit{find}](X,\mathcal{E})},$
$\mathcal{A}=\mathcal{A}^*\cup\mathcal{E},\ S_{13}[*.\mathit{find}](\mathcal{A}^*\cup X),\ \underline{S_{14}[*.\mathit{find}](\mathcal{R}^*,X)},\ \underline{\mathit{lhs}(X)\subseteq\mathcal{R}},$
$\underline{S_{15}[\mathcal{S},*.\mathit{find}](\mathcal{E},X)},\ \underline{S_{16}[\mathcal{S},*.\mathit{find}](\mathcal{R}^*,\mathcal{E},X,\mathcal{S}^*)},\ \underline{S_{17}(\mathcal{E},X)},\ e\in\mathcal{E},\ e\notin X$
77. $e[1].\mathit{find}$ `:=` $e[2]$; [X`:=`$X\cup\{e\}$; `IF` $\neg S_8(e[1])$ `THEN` $\mathcal{S}$`:=`$\{e\}(\mathcal{S})\cup\{e\}$;]
78. $\underline{G},\ \underline{P_\subseteq(\mathcal{F},\mathit{lhs}(\mathcal{S}))},\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_{13}(\mathcal{E}),\ \mathcal{E}\subseteq\Phi,\ \underline{P_{19}[*.\mathit{find}](X,\mathcal{E})},$
$\mathcal{A}=\mathcal{A}^*\cup\mathcal{E},\ \underline{S_{13}[*.\mathit{find}](\mathcal{A}^*\cup X)},\ \underline{S_{14}[*.\mathit{find}](\mathcal{R}^*,X)},\ \underline{\mathit{lhs}(X)\subseteq\mathcal{R}},$
$\underline{S_{15}[\mathcal{S},*.\mathit{find}](\mathcal{E},X)},\ \underline{S_{16}[\mathcal{S},*.\mathit{find}](\mathcal{R}^*,\mathcal{E},X,\mathcal{S}^*)},\ \underline{S_{17}(\mathcal{E},X)}$
79. `END`

80. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{S_3[\mathcal{A},\mathcal{I},*.find]},\ lhs(\mathcal{E}) \subseteq \mathcal{R},$
$\underline{S_{16}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E},\mathcal{E},\mathcal{S}^*)},\ \underline{S_{18}[\mathcal{S},*.find](\mathcal{R}^*)},\ \underline{S_{20}[\mathcal{S}](\mathcal{E})}$
81. $[\ X\ :=\ \emptyset;\ ]$
82. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ lhs(\mathcal{E}) \subseteq \mathcal{R},$
$S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ \underline{S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E}-X,\mathcal{S}^*)},\ S_{20}[\mathcal{S}](\mathcal{E})$
83. `FOREACH` $e \in \mathcal{E}$ `DO BEGIN`
84. $G, P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})), P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}], S_3[\mathcal{A},\mathcal{I},*.find], \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ e \in \mathcal{E},$
$\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ \mathcal{I} \vee S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E}-X,\mathcal{S}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ e \notin X$
85. $\mathcal{L}$ `:=` $e[1].notify$; $[\ \mathcal{U}\ :=\ \emptyset;\ ]$
86. $G, P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})), P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}], S_3[\mathcal{A},\mathcal{I},*.find], \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ e \in \mathcal{E},$
$\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ \underline{S_{21}[*.find](\mathcal{L},e[1])},\ e \notin X,$
$\underline{S_{22}[\mathcal{I},\mathcal{S},*.find](e,\mathcal{R}^*,\mathcal{E}-X,\mathcal{L}-\mathcal{U},\mathcal{S}^*)}$
87. `FOREACH` $(i,d) \in \mathcal{L}$ `DO BEGIN`
88. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),$
$\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ S_{21}[*.find](\mathcal{L},e[1]),\ e \in \mathcal{E},\ e \notin X,$
$S_{22}[\mathcal{I},\mathcal{S},*.find](e,\mathcal{R}^*,\mathcal{E}-X,\mathcal{L}-\mathcal{U},\mathcal{S}^*),\ (i,d) \in \mathcal{L},\ (i,d) \notin \mathcal{U}$
89. $\texttt{TheoryUpdate}_i(e,d)$; $[\ \mathcal{U}\ :=\ \mathcal{U}\ \cup\ \{(i,d)\};\ ]$
90. $\underline{G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),}$
$\underline{\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ S_{21}[*.find](\mathcal{L},e[1]),\ e \in \mathcal{E},\ e \notin X,}$
$\underline{S_{22}[\mathcal{I},\mathcal{S},*.find](e,\mathcal{R}^*,\mathcal{E}-X,\mathcal{L}-\mathcal{U},\mathcal{S}^*)}$
91. `END` $[\ X\ :=\ X\ \cup\ \{e\};\ ]$
92. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),$
$\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ \underline{\mathcal{I} \vee S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E}-X,\mathcal{S}^*)},\ S_{20}[\mathcal{S}](\mathcal{E})$
93. `END`
94. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \underline{\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})},$
$\mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ \underline{\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})}$
95. `END`
96. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),$
$\mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ \mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$
97. `END AssertEqualities`
98. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),$
$\mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ \mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$

99. $G,\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathit{fr}(e),\ P_{11}(e),\ P_{20}[\mathcal{H}](\Phi, e),\ S_4[\mathcal{S}, *.\mathit{find}](e)$
100. `AssertFormula(`e`)` $[\ Z$ `:=` $\emptyset$`;` G_{10}^{ok} `:= FALSE; ]`
101. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(e) \subseteq \mathcal{V}',\ \mathit{fr}(e),\ P_{11}(e),\ \underline{P_{17}[\Phi](\Phi')},\ P_{20}[\mathcal{H}](\Phi', e),$ $\underline{P_{21}[*.\mathit{find}]},\ \underline{P_{38}[\mathcal{A}, \Lambda_*](e)},\ S_4[\mathcal{S}, *.\mathit{find}](e)$
102. `FOREACH maximal sub-term` t `of` e `DO BEGIN`
103. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R}),\ \mathit{free}(e) \subseteq \mathcal{V}',\ \mathit{fr}(e),\ P_{11}(e),$ $P_{17}[\Phi](\Phi'),\ P_{20}[\mathcal{H}](\Phi', e),\ P_{21}[*.\mathit{find}],\ \underline{P_{22}[*.\mathit{find}](Z)},\ \underline{P_{23}(t)},\ \underline{P_{24}[\Lambda_*](Z, e)},$ $P_{38}[\mathcal{A}, \Lambda_*](e),\ S_4[\mathcal{S}, *.\mathit{find}](e),\ \underline{\mathcal{M} \subseteq \mathcal{R}},\ t \lhd e$
104. `SetupTerm(`t`,`$\mathcal{T}(e)$`);` $[\ Z$ `:=` $Z \cup \{t\}$`; ]`
105. $\underline{G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R}),\ \mathit{free}(e) \subseteq \mathcal{V}',\ \mathit{fr}(e),\ P_{11}(e),}$ $\underline{P_{17}[\Phi](\Phi'),\ P_{20}[\mathcal{H}](\Phi', e),\ P_{21}[*.\mathit{find}],\ P_{22}[*.\mathit{find}](Z),\ P_{24}[\Lambda_*](Z, e),}$ $\underline{P_{38}[\mathcal{A}, \Lambda_*](e),\ S_4[\mathcal{S}, *.\mathit{find}](e),\ \mathcal{M} \subseteq \mathcal{R}}$
106. `END`
107. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R}),\ \mathit{free}(e) \subseteq \mathcal{V}',\ P_{11}(e),\ P_{17}[\Phi](\Phi'),$ $P_{20}[\mathcal{H}](\Phi', e),\ P_{21}[*.\mathit{find}],\ \underline{P_{24}[\Lambda_*](e, e)},\ \underline{P_{25}[*.\mathit{find}](e)},\ P_{38}[\mathcal{A}, \Lambda_*](e),$ $\mathcal{M} \subseteq \mathcal{R}$
108. $[\ \mathcal{A}_{\mathcal{T}(e)}$ `:=` $\mathcal{A}_{\mathcal{T}(e)} \cup \{e\}$`;` G_{10}^{ok} `:= TRUE; ]`
109. $\underline{G},\ P_{=}(\mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R}),\ \mathit{free}(e) \subseteq \mathcal{V}',\ \underline{P_{17}[\Phi](\Phi' \cup \{e\})},$ $P_{20}[\mathcal{H}](\Phi', e),\ P_{21}[*.\mathit{find}],\ P_{25}[*.\mathit{find}](e),\ e \in \Phi,\ \underline{\mathcal{A} = \mathcal{A}' \cup \{e\}},\ \mathcal{M} \subseteq \mathcal{R}$
110. `TheoryAssert`$_{\mathcal{T}(e)}$`(`e`);`
111. $\underline{G,\ P_{=}(\mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R}),\ P_{17}[\Phi](\Phi' \cup \{e\}),\ P_{21}[*.\mathit{find}],}$ $\underline{P_{25}[*.\mathit{find}](e),\ e \in \Phi,\ \mathcal{A} = \mathcal{A}' \cup \{e\},\ \mathcal{M} \subseteq \mathcal{R}}$
112. `END AssertFormula`
113. $G,\ P_{=}(\mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R}),\ P_{17}[\Phi](\Phi' \cup \{e\}),\ P_{21}[*.\mathit{find}],$ $P_{25}[*.\mathit{find}](e),\ e \in \Phi,\ \mathcal{A} = \mathcal{A}' \cup \{e\},\ \mathcal{M} \subseteq \mathcal{R}$

114. $G,\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},\ S_4[\mathcal{S}, *.find](t)$
115. `SetupTerm(`t`,`i`)`
116. $G,\ P_{=}(all),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},\ \underline{P_{17}[\Phi](\Phi')},\ S_4[\mathcal{S}, *.find](t)$
117. `IF` $\mathcal{T}(t) \neq i$ `THEN BEGIN`
118. $G,\ P_{=}(all),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},\ P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,$ $S_4[\mathcal{S}, *.find](t)$
119. $[\ \Lambda_{\mathcal{T}(t)}\ :=\ \Lambda_{\mathcal{T}(t)}\ \cup\ \{t\};\ ]$
120. $\underline{G},\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},$ $P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ \underline{P_{39}[\Lambda_*](t)},\ \underline{P_{42}[\Lambda_*](t, i)},\ S_4[\mathcal{S}, *.find](t)$
121. `TheoryAddSharedTerm`$_{\mathcal{T}(t)}$`(`t`);`
122. $\underline{G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},}$ $\underline{P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t, i),\ S_4[\mathcal{S}, *.find](t)}$
123. $[\ \Lambda_i\ :=\ \Lambda_i\ \cup\ \{t\};\ ]$
124. $\underline{G},\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t), free(t) \subseteq \mathcal{V}, t \in \Lambda_i,$ $P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ \underline{P_{39}[\Lambda_*](t)},\ \underline{P_{42}[\Lambda_*](t, i)},\ S_4[\mathcal{S}, *.find](t)$
125. `TheoryAddSharedTerm`$_i$`(`t`);`
126. $\underline{G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t), free(t) \subseteq \mathcal{V}, t \in \Lambda_i,}$ $\underline{P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t, i),\ S_4[\mathcal{S}, *.find](t)}$
127. `END`
128. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},$ $P_{17}[\Phi](\Phi'),\ \underline{P_{26}[\Lambda_*](t, i)},\ \underline{P_{39}[\Lambda_*](t)},\ \underline{P_{42}[\Lambda_*](t, i)},\ S_4[\mathcal{S}, *.find](t)$
129. `IF HasFind(`t`) THEN BEGIN`
130. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{17}[\Phi](\Phi'),\ hf(t),$ $\underline{P_{24}[\Lambda_*](t, t)},\ P_{26}[\Lambda_*](t, i),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t, i),\ S_4[\mathcal{S}, *.find](t)$
131. `RETURN;`
132. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*, \underline{\mathcal{F}, \mathcal{R}}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t, t),$ $P_{26}[\Lambda_*](t, i),\ \underline{P_{27}[*.find](t)},\ \underline{P_{28}[*.find](t)},\ \underline{t.find \equiv t},\ P_{39}[\Lambda_*](t),$ $P_{42}[\Lambda_*](t, i),\ \underline{\mathcal{M} \subseteq \mathcal{R}}$
133. `END`
134. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ free(t) \subseteq \mathcal{V},\ fr(t),\ P_{17}[\Phi](\Phi'),$ $P_{23}(t),\ P_{26}[\Lambda_*](t, i),\ \neg hf(t),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t, i),\ S_4[\mathcal{S}, *.find](t)$
135. `FOR` k `:= 1 TO` $Arity(t)$ `DO BEGIN`
136. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*, \underline{\mathcal{F}, \mathcal{R}}),\ free(t) \subseteq \mathcal{V},\ P_{17}[\Phi](\Phi'),\ P_{23}(t),$ $\neg hf(t),\ P_{26}[\Lambda_*](t, i),\ \underline{P_{27}[*.find](t)},\ \underline{P_{28}[*.find](t)},\ \underline{P_{29}[*.find](t, k)},$ $\underline{P_{30}[*.find](t, Arity(t) + 1)},\ \underline{P_{32}[\Lambda_*](t, k)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t, i),$ $S_4[\mathcal{S}, *.find](t),\ \underline{\mathcal{M} \subseteq \mathcal{R}}$
137. `SetupTerm(`$t[k]$`,`$\mathcal{T}(t)$`);`
138. $\underline{G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*, \mathcal{F}, \mathcal{R}),\ free(t) \subseteq \mathcal{V},\ P_{17}[\Phi](\Phi'),\ P_{23}(t),}$ $\underline{\neg hf(t),\ P_{26}[\Lambda_*](t, i),\ P_{27}[*.find](t),\ P_{28}[*.find](t),\ P_{29}[*.find](t, k + 1),}$ $\underline{P_{30}[*.find](t, Arity(t) + 1),\ P_{32}[\Lambda_*](t, k + 1),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t, i),}$ $\underline{S_4[\mathcal{S}, *.find](t),\ \mathcal{M} \subseteq \mathcal{R}}$
139. `END`

140. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ free(t)\subseteq\mathcal{V},\ P_{17}[\Phi](\Phi'),\ P_{23}(t),$
$P_{24}[\Lambda_*](t,t),\ P_{26}[\Lambda_*](t,i),\ \neg hf(t),\ P_{27}[*.find](t),\ P_{28}[*.find](t),\ P_{31}[*.find](t),$
$P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.find](t),\ \mathcal{M}\subseteq\mathcal{R}$
141. $t.find$ `:=` t`;` [G_{16}^{ok} `:= FALSE;`]
142. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$
$P_{26}[\Lambda_*](t,i),\ P_{27}[*.find](t),\ P_{28}[*.find](t),\ t.find\equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$
$\mathcal{M}\subseteq\mathcal{R},\ S_{23}[*.find,*.notify](t)$
143. `TheorySetup`$_{\mathcal{T}(t)}$`(`t`);` [G_{16}^{ok} `:= TRUE;`]
144. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$
$P_{26}[\Lambda_*](t,i),\ P_{27}[*.find](t),\ P_{28}[*.find](t),\ t.find\equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$
$\mathcal{M}\subseteq\mathcal{R}$
145. `END SetupTerm`
146. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$
$P_{26}[\Lambda_*](t,i),\ P_{27}[*.find](t),\ P_{28}[*.find](t),\ t.find\equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$
$\mathcal{M}\subseteq\mathcal{R}$

147. $G,\ free(e)\subseteq\mathcal{V}$
148. `Simplify(`e`)`
149. $G,\ P_{=}(all),\ free(e)\subseteq\mathcal{V},\ \mathcal{T}\cup\Phi\models e'\simeq e$
150. `IF HasFind(`e`) THEN BEGIN`
151. $G,\ P_{=}(all),\ \mathcal{T}\cup\Phi\models e'\simeq e,\ hf(e)$
152. `RETURN Find(`e`);`
153. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V}),\ free(retval)\subseteq\mathcal{V},\ fr(retval),$
$\mathcal{T}\cup\Phi\models e'\simeq retval,\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S},*.find](retval),\ S_5(retval)$
154. `END`
155. $G,\ P_{=}(all),\ free(e)\subseteq\mathcal{V},\ \mathcal{T}\cup\Phi\models e'\simeq e,\ \neg hf(e)$
156. `FOR` k `:= 1 to` $Arity(e)$ `DO BEGIN`
157. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V}),\ free(e)\subseteq\mathcal{V},\ \mathcal{T}\cup\Phi\models e'\simeq e,$
$P_{17}[\Phi](\Phi'),\ P_{30}[*.find](e,k),\ S_{24}[\mathcal{S},*.find](e,k)$
158. $e[k]$ `:= Simplify(`$e[k]$`);`
159. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V}),\ free(e)\subseteq\mathcal{V},\ \mathcal{T}\cup\Phi\models e'\simeq e,$
$P_{17}[\Phi](\Phi'),\ P_{30}[*.find](e,k+1),\ S_{24}[\mathcal{S},*.find](e,k+1)$
160. `END`
161. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V}),\ free(e)\subseteq\mathcal{V},\ \mathcal{T}\cup\Phi\models e'\simeq e,$
$P_{17}[\Phi](\Phi'),\ P_{30}[*.find](e,Arity(e)+1),\ S_{24}[\mathcal{S},*.find](e,Arity(e)+1)$
162. `RETURN Rewrite(`e`);`
163. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V}),\ free(retval)\subseteq\mathcal{V},\ fr(retval),$
$\mathcal{T}\cup\Phi\models e'\simeq retval,\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S},*.find](retval),\ S_5(retval)$
164. `END Simplify`
165. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V}),\ free(retval)\subseteq\mathcal{V},\ fr(retval),$
$\mathcal{T}\cup\Phi\models e'\simeq retval,\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S},*.find](retval),\ S_5(retval)$

166. $G,\ free(e) \subseteq \mathcal{V},\ fr(e) \vee hf(e),\ S_{25}[\mathcal{S}, *.find](e)$
167. `Rewrite(`e`)`
168. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e) \vee hf(e),\ S_{25}[\mathcal{S}, *.find](e)$
169. `IF HasFind(`e`) THEN BEGIN`
170. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ hf(e)$
171. `RETURN Find(`e`);`
172. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),\ \mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S}, *.find](retval),\ S_5(retval)$
173. `END`
174. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_{25}[\mathcal{S}, *.find](e)$
175. e^* `:= OpRewrite(`e`);`
176. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,\ fr(e^*),\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S}, *.find](e, e^*),\ S_{27}(e, e^*)$
177. `IF` $e \not\equiv e^*$ `THEN BEGIN`
178. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,\ fr(e^*),\ P_{17}[\Phi](\Phi'),\ S_{25}[\mathcal{S}, *.find](e)$
179. e^* `:= Rewrite(`e^*`);`
180. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,\ fr(e^*),\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S}, *.find](e^*),\ S_5(e^*)$
181. `END`
182. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,\ fr(e^*),\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S}, *.find](e^*),\ S_5(e^*)$
183. `RETURN` e^*`;`
184. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),\ \mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S}, *.find](retval),\ S_5(retval)$
185. `END Rewrite`
186. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),\ \mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S}, *.find](retval),\ S_5(retval)$

187. $G,\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_{25}[\mathcal{S}, *.find](e)$
188. `OpRewrite(`e`)`
189. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_{25}[\mathcal{S}, *.find](e)$
190. `IF` $Op(e) =$ '$\neg$' `THEN BEGIN`
191. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $S_4[\mathcal{S}, *.find](e),\ S_5(e)$
192. `RETURN RewriteNegation(`e`);`
193. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),\ \mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S}, *.find](e, retval),\ S_{27}(e, retval)$
194. `END`

195. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_{25}[\mathcal{S}, *.find](e)$
196. IF $Op(e) =$ '=' AND $e[1] \equiv e[2]$ THEN BEGIN
197. $G,\ P_{=}(all),\ Op(e) =$ '=', $e[1] \equiv e[2]$
198. RETURN *true*;
199. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),$ $\mathcal{T} \cup \Phi \models e \simeq retval,\ \underline{P_{17}[\Phi](\Phi')},\ \underline{S_{26}[\mathcal{S}, *.find](e, retval)},\ \underline{S_{27}(e, retval)}$
200. END
201. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ \underline{S_5(e)},\ S_{25}[\mathcal{S}, *.find](e)$
202. IF e is a term or an atomic formula THEN BEGIN
203. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ \underline{P_{33}(e)},\ S_5(e),\ S_{25}[\mathcal{S}, *.find](e)$
204. RETURN TheoryRewrite$_{\mathcal{T}(e)}$(e);
205. $\underline{G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),}$ $\underline{\mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S}, *.find](e, retval),\ S_{27}(e, retval)}$
206. END
207. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ \underline{S_4[\mathcal{S}, *.find](e)},\ S_5(e)$
208. RETURN e;
209. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),$ $\mathcal{T} \cup \Phi \models e \simeq retval,\ \underline{P_{17}[\Phi](\Phi')},\ \underline{S_{26}[\mathcal{S}, *.find](e, retval)},\ \underline{S_{27}(e, retval)}$
210. END OpRewrite
211. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),$ $\mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S}, *.find](e, retval),\ S_{27}(e, retval)$

212. $G,\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $S_4[\mathcal{S}, *.find](e)$
213. RewriteNegation(e)
214. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $S_4[\mathcal{S}, *.find](e)$
215. IF $e[1] \equiv$ *true* THEN BEGIN
216. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $e[1] \equiv true$
217. RETURN *false*;
218. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),$ $\underline{S_4[\mathcal{S}, *.find](retval)}$
219. END
220. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $S_4[\mathcal{S}, *.find](e)$
221. IF $e[1] \equiv$ *false* THEN BEGIN
222. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $e[1] \equiv false$
223. RETURN *true*;
224. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),$ $\underline{S_4[\mathcal{S}, *.find](retval)}$
225. END

226. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) = \text{'}\neg\text{'},\ S_4[\mathcal{S}, *.find](e)$
227. IF $Op(e[1]) \equiv \text{'}\neg\text{'}$ THEN BEGIN
228. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) = \text{'}\neg\text{'},\ Op(e[1]) = \text{'}\neg\text{'},\ S_4[\mathcal{S}, *.find](e)$
229. RETURN $e[1][1]$;
230. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),\ S_4[\mathcal{S}, *.find](retval)$
231. END
232. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_4[\mathcal{S}, *.find](e)$
233. RETURN e;
234. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),\ S_4[\mathcal{S}, *.find](retval)$
235. END RewriteNegation
236. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),\ S_4[\mathcal{S}, *.find](retval)$

237. $G,\ hf(t)$
238. Find(t)
239. $G,\ P_{=}(all),\ hf(t)$
240. IF $t.find \equiv\ t$ THEN BEGIN
241. $G,\ P_{=}(all),\ t.find \equiv t$
242. RETURN t;
243. $G,\ P_{=}(all),\ retval.find \equiv retval,\ t \sim retval$
244. END ELSE BEGIN
245. $G,\ P_{=}(all),\ hf(t)$
246. RETURN Find($t.find$);
247. $G,\ P_{=}(all),\ retval.find \equiv retval,\ t \sim retval$
248. END
249. *unreachable*
250. END Find
251. $G,\ P_{=}(all),\ retval.find \equiv retval,\ t \sim retval$

A.4.2 API for Theory-Specific Code

252. G, $\mathit{free}(e) \subseteq \mathcal{V}$
253. `TheoryAddSharedTerm`$_i$`(`e`)`
254. G, $P_{34}[\mathit{all}](i)$

255. G,
256. `TheoryAssert`$_i$`(`e`)`
257. G, $P_{34}[\mathit{all}](i)$

258. G, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, convex, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$
259. `TheoryCheckSat`$_i$`()`
260. G, $P_{=}(\mathit{all} - \{\mathcal{I}, \mathcal{N}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](i)$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$

261. G, $\mathit{free}(e) \subseteq \mathcal{V}$, $\mathit{fr}(e)$, $P_{33}(e)$, $S_{25}[\mathcal{S}, *.\mathit{find}](e)$
262. `TheoryRewrite`$_i$`(`e`)`
263. G, $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$, $\mathit{fr}(\mathit{retval})$, $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$, $P_{34}[\mathit{all}](i)$, $S_{26}[\mathcal{S}, *.\mathit{find}](e, \mathit{retval})$

264. G, $\mathit{hf}(e)$
265. `TheorySetup`$_i$`(`e`)`
266. G, $P_{34}[\mathit{all}](i)$

267. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi, \mathcal{H}](e)$, $\mathit{free}(e) \subseteq \mathcal{V}$, $\mathit{fr}(e)$, $P_{11}(e)$, $\mathit{Op}(e) = \text{`}=\text{'}$, $e[1] \not\equiv e[2]$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_4[\mathcal{S}, *.\mathit{find}](e)$
268. `TheorySolve(`e`)`
269. G, $P_{=}(\mathcal{H}, \mathcal{I}, \mathcal{N}, *.\mathit{find})$, $P_{\subseteq}(\Phi)$, $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$, $P_{14}[*.\mathit{find}](\mathit{retval})$, $P_{17}[\Phi](\Phi')$, $P_{35}[\Phi](e, \mathit{retval})$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_7[\mathcal{S}, *.\mathit{find}](\mathit{retval})$

270. G, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_{28}[*.\mathit{find}](i, d)$
271. `TheoryUpdate`$_i$`(`e`,`d`)`
272. G, $P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})$, $\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$, $\mathcal{I}' \to \mathcal{I}$

A.4.3 Theory-Specific Code for a Nelson-Oppen Theory $\mathcal{T}_i$

273. G, $free(e) \subseteq \mathcal{V}$
274. `TheoryAddSharedTerm`$_i$`(`e`)`
275. G, $\underline{P_{34}[all](i)}$

276. G,
277. `TheoryAssert`$_i$`(`e`)`
278. G, $\underline{P_{34}[all](i)}$

279. G, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, $convex$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$
280. `TheoryCheckSat`$_i$`()`
281. G, $P_=(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, $convex$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$
282. `IF` $\neg Sat_i(\Phi_i \cup E_{\sim_i})$ `THEN BEGIN`
283. G, $P_=(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, $convex$, $\underline{\neg P_{36}[\Phi, \Lambda_*, *.find](i)}$
284. $\mathcal{I}$ `:= TRUE;`
285. $\underline{G}$, $P_=(all - \{\mathcal{I}\})$, $\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$, $\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)}$, $\underline{S_2[\mathcal{I}, \mathcal{S}, *.find]}$, $\underline{S_3[\mathcal{A}, \mathcal{I}, *.find]}$
286. `END ELSE IF` $\neg Sat_i(\Phi_i \cup Ar_{\sim_i})$ `THEN BEGIN`
287. G, $P_=(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, $convex$, $\underline{\neg P_7[\Phi, \Lambda_*, *.find](i)}$, $\underline{P_{36}[\Phi, \Lambda_*, *.find](i)}$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$
288. `Choose` $\Delta \subseteq D_{\sim_i}$ `such that` $\neg Sat_i(\Phi_i \cup E_{\sim_i} \cup \Delta)$`;`
289. G, $P_=(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, $convex$, $\neg P_7[\Phi, \Lambda_*, *.find](i)$, $P_{36}[\Phi, \Lambda_*, *.find](i)$, $\underline{P_{37}[\Phi, \Lambda_*, *.find](i, \neg\Delta)}$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$
290. $\mathcal{Q}$ `:=` $\{\neg\Delta\}$`;`
291. $\underline{G}$, $P_=(all - \{\mathcal{I}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)}$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$
292. `END`
293. G, $P_=(all - \{\mathcal{I}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)}$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$
294. `END TheoryCheckSat`$_i$
295. G, $P_=(all - \{\mathcal{I}, \mathcal{N}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

296. G, $\mathit{free}(e) \subseteq \mathcal{V}$, $\mathit{fr}(e)$, $P_{33}(e)$, $S_{25}[\mathcal{S}, *.\mathit{find}](e)$
297. `TheoryRewrite`$_i$`(`e`)`
298. `  RETURN` e`;`
299. G, $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$, $\mathit{fr}(\mathit{retval})$, $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$, $\underline{P_{34}[\mathit{all}](i)}$, $\underline{S_{26}[\mathcal{S}, *.\mathit{find}](e, \mathit{retval})}$

300. G, $\mathit{hf}(e)$
301. `TheorySetup`$_i$`(`e`)`
302. G, $\underline{P_{34}[\mathit{all}](i)}$

303. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi, \mathcal{H}](e)$, $\mathit{free}(e) \subseteq \mathcal{V}$, $\mathit{fr}(e)$, $P_{11}(e)$, $\mathit{Op}(e) = \text{`}=\text{'}$, $e[1] \not\equiv e[2]$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_4[\mathcal{S}, *.\mathit{find}](e)$
304. `TheorySolve(`e`)`
305. `  RETURN` $\{e\}$`;`
306. G, $P_=(\mathcal{H}, \mathcal{I}, \mathcal{N}, *.\mathit{find})$, $P_\subseteq(\Phi)$, $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$, $\underline{P_{14}[*.\mathit{find}](\mathit{retval})}$, $\underline{P_{17}[\Phi](\Phi')}$, $\underline{P_{35}[\Phi](e, \mathit{retval})}$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\underline{S_7[\mathcal{S}, *.\mathit{find}](\mathit{retval})}$

307. G, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_{28}[*.\mathit{find}](i, d)$
308. `TheoryUpdate`$_i$`(`e`,`d`)`
309. G, $P_\subseteq(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\underline{\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})}$, $\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$, $\mathcal{I}' \rightarrow \mathcal{I}$

A.4.4 Theory-Specific Code for Shostak Theory $\mathcal{T}_\chi$

310. G, $\mathit{free}(e) \subseteq \mathcal{V}$
311. `TheoryAddSharedTerm`$_i$`(`e`)`
312. G, $\underline{P_{34}[\mathit{all}](i)}$

313. G,
314. `TheoryAssert`$_i$`(`e`)`
315. G, $\underline{P_{34}[\mathit{all}](i)}$

316. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \mathcal{Q}=\emptyset,\ \neg\mathcal{I},\ convex,\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
317. `TheoryCheckSat`$_\chi$`()` [X := $\emptyset$;]
318. $G,\ P_=(all),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
319. `FOREACH` e `in` $\mathcal{A}_\chi$ `DO BEGIN`
320. $G,\ P_=(all),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ e\in\mathcal{A}_\chi,\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find],$ $\underline{S_{29}[\mathcal{I},\mathcal{S}](X,\chi)}$
321. `IF` $Op(e)=$'$\neg$' `AND Find(`$e[1][1]$`)` $\equiv$ `Find(`$e[1][2]$`) THEN BEGIN`
322. $\underline{G,\ P_=(all),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ Op(e[1])=\text{`}=\text{'},\ Op(e)=\text{'}\neg\text{'},}$ $\underline{e[1][1]\sim e[1][2],\ e\in\mathcal{A}_\chi,\ S_2[\mathcal{I},\mathcal{S},*.find]}$
323. $\mathcal{I}$:= `TRUE; RETURN;`
324. $\underline{G},\ P_=(all-\{\mathcal{I},\mathcal{N},\mathcal{Q}\}),\ \underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]},\ \underline{P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](\chi)},$ $\underline{S_2[\mathcal{I},\mathcal{S},*.find]},\ \underline{S_3[\mathcal{A},\mathcal{I},*.find]}$
325. `END` [X := $X\ \cup\ \{e\}$;]
326. $G,\ P_=(all),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find],$ $\underline{S_{29}[\mathcal{I},\mathcal{S}](X,\chi)}$
327. `END`
328. $G,\ P_=(all-\{\mathcal{I},\mathcal{N},\mathcal{Q}\}),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](\chi)},$ $S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
329. `END` `TheoryCheckSat`$_\chi$
330. $G,\ P_=(all-\{\mathcal{I},\mathcal{N},\mathcal{Q}\}),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.find](\chi),$ $S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$

331. $G,\ free(e)\subseteq\mathcal{V},\ fr(e),\ P_{33}(e),\ S_{25}[\mathcal{S},*.find](e)$
332. `TheoryRewrite`$_\chi(e)$
333. $G,\ P_=(all),\ free(e)\subseteq\mathcal{V},\ fr(e),\ P_{33}(e),\ S_{25}[\mathcal{S},*.find](e)$
334. `IF` e `is not a term THEN BEGIN`
335. $G,\ P_=(all),\ free(e)\subseteq\mathcal{V},\ fr(e),\ P_{33}(e),\ \underline{\neg P_{23}(e)},\ \underline{S_4[\mathcal{S},*.find](e)}$
336. `RETURN` e;
337. $G,\ \mathcal{T}\cup\Phi\models e\simeq retval,\ fr(retval),\ free(retval)\subseteq\mathcal{V},\ \underline{P_{34}[all](\chi)},$ $\underline{S_{26}[\mathcal{S},*.find](e,retval)},\ \underline{S_{42}[\mathcal{S}](e,retval)}$
338. `END`
339. $G,\ P_=(all),\ free(e)\subseteq\mathcal{V},\ fr(e),\ \underline{P_{23}(e)},\ S_{25}[\mathcal{S},*.find](e)$
340. e^* := `RewriteHelper(`e`);`
341. $\underline{G,\ P_=(all),\ free(e^*)\subseteq\mathcal{V},\ P_{23}(e^*),\ \mathcal{T}\cup\Phi\models e\simeq e^*,\ S_{30}[*.find](e^*),}$ $\underline{S_{31}[\mathcal{S},*.find](e^*),\ S_{32}[\mathcal{S}](e^*),\ S_{33}[\mathcal{S}](e^*,e)}$
342. `RETURN` $canon_\chi(e^*)$;
343. $G,\ \underline{\mathcal{T}\cup\Phi\models e\simeq retval},\ \underline{fr(retval)},\ \underline{free(retval)\subseteq\mathcal{V}},\ \underline{P_{34}[all](\chi)},$ $\underline{S_{26}[\mathcal{S},*.find](e,retval)},\ \underline{S_{42}[\mathcal{S}](e,retval)}$
344. `END` `TheoryRewrite`$_\chi$
345. $G,\ \mathcal{T}\cup\Phi\models e\simeq retval,\ fr(retval),\ free(retval)\subseteq\mathcal{V},\ P_{34}[all](\chi),$ $S_{26}[\mathcal{S},*.find](e,retval),\ S_{42}[\mathcal{S}](e,retval)$

346. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e)$, $\mathit{free}(e)\subseteq\mathcal{V}$, $\mathit{fr}(e)$, $P_{11}(e)$, $\mathit{Op}(e)=\text{`='}$, $e[1]\not\equiv e[2]$, $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $S_4[\mathcal{S},*.\mathit{find}](e)$
347. `TheorySolve(`e`)`
348. G, $P_=(\mathit{all})$, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e)$, $\mathit{free}(e)\subseteq\mathcal{V}$, $\mathit{fr}(e)$, $P_{11}(e)$, $\mathit{Op}(e)=\text{`='}$, $e[1]\not\equiv e[2]$, $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $S_4[\mathcal{S},*.\mathit{find}](e)$
349. `RETURN` $\mathit{solve}_\chi(e)$; [$\mathcal{V}$:= $\mathcal{V}\cup\mathit{free}(\mathit{retval})$]
350. G, $P_=(\Phi,\mathcal{H},\mathcal{I},\mathcal{N},*.\mathit{find})$, $\mathit{free}(\mathit{retval})\subseteq\mathcal{V}$, $\underline{P_{14}[*.\mathit{find}](\mathit{retval})}$, $\underline{P_{17}[\Phi](\Phi')}$, $\underline{P_{35}[\Phi](e,\mathit{retval})}$, $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $\underline{S_7[\mathcal{S},*.\mathit{find}](\mathit{retval})}$
351. `END TheorySolve`
352. G, $P_=(\mathcal{H},\mathcal{I},\mathcal{N},*.\mathit{find})$, $P_\subseteq(\Phi)$, $\mathit{free}(\mathit{retval})\subseteq\mathcal{V}$, $P_{14}[*.\mathit{find}](\mathit{retval})$, $P_{17}[\Phi](\Phi')$, $P_{35}[\Phi](e,\mathit{retval})$, $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $S_7[\mathcal{S},*.\mathit{find}](\mathit{retval})$

353. G, $\mathit{hf}(e)$
354. `TheorySetup`$_\chi$`(`e`)`
355. G, $P_=(\mathit{all})$, $\mathit{hf}(e)$
356. `IF` e `is a compound` χ`-term THEN BEGIN` [Z := $\emptyset$;]
357. G, $P_=(\mathit{all})$, $\mathit{hf}(e)$, $\underline{S_8(e)}$
358. `FOREACH` $c\in\delta_\chi(e)$ `DO BEGIN`
359. G, $P_=(\mathit{all}-\{*.\mathit{notify}\})$, $P_\subseteq(*.\mathit{notify})$, $\mathit{hf}(e)$, $S_8(e)$, $\underline{S_{36}[*.\mathit{notify}](Z)}$, $c\in\delta_\chi(e)$
360. $c.\mathit{notify}$:= $c.\mathit{notify}\cup\{(\chi,e)\}$; [Z := $Z\cup\{c\}$;]
361. $\underline{G}$, $P_=(\mathit{all}-\{*.\mathit{notify}\})$, $P_\subseteq(*.\mathit{notify})$, $\mathit{hf}(e)$, $S_8(e)$, $\underline{S_{36}[*.\mathit{notify}](Z)}$
362. `END`
363. G, $\underline{P_{34}[\mathit{all}](\chi)}$, $\underline{S_{35}[*.\mathit{notify}](e)}$
364. `END`
365. G, $\underline{P_{34}[\mathit{all}](\chi)}$, $\underline{S_{35}[*.\mathit{notify}](e)}$
366. `END TheorySetup`$_\chi$
367. G, $P_{34}[\mathit{all}](\chi)$, $S_{35}[*.\mathit{notify}](e)$

368. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ hf(d),\ S_3[\mathcal{A},\mathcal{I},*.find],\ S_8(d),\ S_9[*.find](d)$
369. TheoryUpdate$_\chi(e,d)$
370. $G,\ P_=(all),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ hf(d),\ S_3[\mathcal{A},\mathcal{I},*.find],\ S_8(d),\ S_9[*.find](d)$
371. IF $\neg\mathcal{I}$ AND Find(d) $\equiv d$ THEN BEGIN
372. $G,\ P_=(all),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \neg\mathcal{I},\ d.find \equiv d,\ S_3[\mathcal{A},\mathcal{I},*.find],\ S_8(d),\ S_9[*.find](d)$
373. d^* := TheoryRewrite$_\chi(d)$;
374. $G,\ P_=(\mathcal{S},*.find),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \neg\mathcal{I},\ \neg\mathcal{I}',\ fr(d^*),\ P_{23}(d^*),\ d.find \equiv d,\ S_1[\mathcal{S}](d^*,d),\ S_3[\mathcal{A},\mathcal{I},*.find],\ S_4[\mathcal{S},*.find](d^*),\ S_8(d),\ S_9[*.find](d)$
375. AssertEqualities($\{d = d^*\}$);
376. $G,\ P_\subseteq(\mathcal{F},lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \neg\mathcal{I}',\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I}\vee(\mathcal{M}\subseteq\mathcal{R}),\ \mathcal{I}\vee(\mathcal{R}'\subseteq\mathcal{R}),\ \mathcal{I}\vee(d\in\mathcal{R})$
377. END
378. $G,\ P_\subseteq(\mathcal{F},lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I}\vee(\mathcal{M}\subseteq\mathcal{R}),\ \mathcal{I}\vee(\mathcal{R}'\subseteq\mathcal{R}),\ \mathcal{I}'\to\mathcal{I},\ S_{34}[\mathcal{I},\mathcal{S},*.find](d)$
379. END TheoryUpdate$_\chi$
380. $G,\ P_\subseteq(\mathcal{F},lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I}\vee(\mathcal{M}\subseteq\mathcal{R}),\ \mathcal{I}\vee(\mathcal{R}'\subseteq\mathcal{R}),\ \mathcal{I}'\to\mathcal{I},\ S_{34}[\mathcal{I},\mathcal{S},*.find](d)$

381. $G,\ free(t)\subseteq\mathcal{V},\ P_{23}(t),\ fr(t)\vee hf(t),\ S_{25}[\mathcal{S},*.find](t)$
382. RewriteHelper(t)
383. $G,\ P_=(all),\ free(t)\subseteq\mathcal{V},\ t'\equiv\ t,\ P_{23}(t),\ fr(t)\vee hf(t),\ S_{25}[\mathcal{S},*.find](t)$
384. IF t is a χ-leaf THEN BEGIN
385. $G,\ P_=(all),\ free(t)\subseteq\mathcal{V},\ t'\equiv\ t,\ P_{23}(t),\ fr(t)\vee hf(t),\ \neg S_8(t),\ S_{25}[\mathcal{S},*.find](t)$
386. IF ¬HasFind(t) OR $t.find\equiv\ t$ THEN BEGIN
387. $G,\ P_=(all),\ free(t)\subseteq\mathcal{V},\ t'\equiv\ t,\ fr(t),\ P_{23}(t),\ \neg S_8(t),\ S_{31}[\mathcal{S},*.find](t),\ S_{32}[\mathcal{S}](t),\ S_{33}[\mathcal{S}](t,t')$
388. RETURN t;
389. $G,\ P_=(all),\ free(retval)\subseteq\mathcal{V},\ \mathcal{T}\cup\Phi\models t'\simeq retval,\ P_{23}(retval),\ S_{30}[*.find](retval),\ S_{31}[\mathcal{S},*.find](retval),\ S_{32}[\mathcal{S}](retval),\ S_{33}[\mathcal{S}](retval,t')$
390. END ELSE BEGIN
391. $G,\ P_=(all),\ free(t)\subseteq\mathcal{V},\ t'\equiv\ t,\ P_{23}(t),\ hf(t),\ S_{25}[\mathcal{S},*.find](t)$
392. t := Find(t);
393. $G,\ P_=(all),\ free(t)\subseteq\mathcal{V},\ t'\sim\ t,\ P_{23}(t),\ fr(t),\ S_{25}[\mathcal{S},*.find](t)$
394. RETURN RewriteHelper(t^*);
395. $G,\ P_=(all),\ free(retval)\subseteq\mathcal{V},\ \mathcal{T}\cup\Phi\models t'\simeq retval,\ P_{23}(retval),\ S_{30}[*.find](retval),\ S_{31}[\mathcal{S},*.find](retval),\ S_{32}[\mathcal{S}](retval),\ S_{33}[\mathcal{S}](retval,t')$
396. END
397. *unreachable*

```
398.   END ELSE BEGIN
399.     G, P_=(all), free(t) ⊆ V, t' ≡ t, P_23(t), fr(t) ∨ hf(t), S_8(t),
         S_25[S, *.find](t)
400.     FOR k := 1 to Arity(t) DO BEGIN
401.       G, P_=(all), free(t) ⊆ V, T ∪ Φ |= t' ≃ t, P_23(t), S_8(t), S_37[*.find](t),
           S_38[*.find](t, k), S_39[S, *.find](t, k), S_40[S, *.find](t, k), S_41[S](t, k),
           Op(t) = Op(t')
402.       t[k] := RewriteHelper(t[k]);
403.       G, P_=(all), free(t) ⊆ V, T ∪ Φ |= t' ≃ t, P_23(t), S_8(t), S_37[*.find](t),
           S_38[*.find](t, k + 1), S_39[S, *.find](t, k + 1), S_40[S, *.find](t, k + 1),
           S_41[S](t, k + 1), Op(t) = Op(t')
404.     END
405.     G, P_=(all), free(t) ⊆ V, T ∪ Φ |= t' ≃ t, P_23(t), S_30[*.find](t),
         S_31[S, *.find](t), S_32[S](t), S_33[S](t, t')
406.     RETURN t;
407.     G, P_=(all), free(retval) ⊆ V, T ∪ Φ |= t' ≃ retval, P_23(retval),
         S_30[*.find](retval), S_31[S, *.find](retval), S_32[S](retval), S_33[S](retval, t')
408.   END
409.   unreachable
410. END RewriteHelper
411. G, P_=(all), free(retval) ⊆ V, T ∪ Φ |= t' ≃ retval, P_23(retval),
     S_30[*.find](retval), S_31[S, *.find](retval), S_32[S](retval), S_33[S](retval, t')
```

A.5 Detailed Proof

Finally, we present a detailed, line-by-line proof of each underlined property listed in the annotated code. For each line, the code together with the properties before (preconditions) and after (postconditions) are reproduced for convenience. Then, for each underlined postcondition, a justification is given.

It is often necessary to refer to the value of state variables before the line is executed. This is done by subscripting the variable with the line number of the preconditions. For example, in the proof for line 2, $\mathcal{H}_0$ refers to the value of the assumption history just *before* entering `AddFact`, while $\mathcal{H}$ refers to the value at line 2, after entering and executing the virtual code which changes $\mathcal{H}$.

A.5.1 Lemmas

We start with a few general-purpose lemmas.

Lemma A.1. *Suppose t is a Σ-term, $\mathcal{S}$ is a set of equations in χ-solved form, and $canon_\chi$ is a generalized canonizer as described in Section A.2.1. Then if $s \equiv canon_\chi(\mathcal{S}(t))$, then $\mathcal{S}(s) \equiv s$.*

Proof. Suppose that $\mathcal{S}(s) \not\equiv s$. Then since $\mathcal{S}$ is in χ-solved form, there must be some $c \in lhs(\mathcal{S})$ such that $c \in \delta_\chi(s)$. Then, by property 3 of *canon*, this means that $c \in \delta_\chi(\mathcal{S}(t))$. But because $\mathcal{S}$ is in χ-solved form, $\delta_\chi(\mathcal{S}(t)) \cap lhs(\mathcal{S}) = \emptyset$ which contradicts the fact that $c \in lhs(\mathcal{S})$. Thus, $\mathcal{S}(s) \equiv s$. □

Lemma A.2. *Suppose $\mathcal{T}_\chi$ is a Shostak theory with signature Σ_χ and canonizer canon, t is a Σ_χ-term, and $\mathcal{S}'$, $\mathcal{S}$, and $\{e\}$ are sets of Σ_χ-equations in solved form such that $\mathcal{S} = \{e\}(\mathcal{S}') \cup \{e\}$. Then $canon(\mathcal{S}(canon(\mathcal{S}'(t)))) \equiv canon(\mathcal{S}(t))$.*

Proof.

$\mathcal{T}_\chi \models t = t$	$\mathcal{T}_\chi$ includes reflexivity.
$\mathcal{T}_\chi \cup \mathcal{S}' \models \mathcal{S}'(t) = t$	$\mathcal{S}'$ in solved form.
$\mathcal{T}_\chi \cup \mathcal{S}' \models canon(\mathcal{S}'(t)) = t$	Properties 1 and 2 of *canon*
$\mathcal{T}_\chi \cup \mathcal{S}' \cup \mathcal{S} \models \mathcal{S}(canon(\mathcal{S}'(t))) = \mathcal{S}(t)$	$\mathcal{S}$ in solved form.
$\mathcal{T}_\chi \cup \mathcal{S} \models \mathcal{S}(canon(\mathcal{S}'(t))) = \mathcal{S}(t)$	$\mathcal{S} \models \mathcal{S}'$.
$\mathcal{T}_\chi \models \mathcal{S}(canon(\mathcal{S}'(t))) = \mathcal{S}(t)$	Proposition 2.1.
$canon(\mathcal{S}(canon(\mathcal{S}'(t)))) \equiv canon(\mathcal{S}(t))$	Property 1 of *canon*.

□

Corollary A.1. *Suppose $\mathcal{T}_\chi$ is a Shostak theory which is part of a combined theory $\mathcal{T}$ with signature Σ, and suppose $canon_\chi$ is a generalized canonizer as described in Section A.2.1. Then, if t is a Σ-term, and $\mathcal{S}_0, \mathcal{S}_1, \ldots \mathcal{S}_n$ are sets of equations in χ-solved form, where for each $i, 1 \leq i \leq n$, there exists e such that $\{e\}$ is in χ-solved form and $\mathcal{S}_i = \{e\}(\mathcal{S}_{i-1}) \cup \{e\}$, then $canon_\chi(\mathcal{S}_n(canon_\chi(\mathcal{S}_0(t)))) \equiv canon_\chi(\mathcal{S}_n(t))$.*

Lemma A.3. *Suppose $\mathcal{T}_\chi$ is a Shostak theory which is part of a combined theory $\mathcal{T}$, and suppose $canon_\chi$ is a generalized canonizer as described in Section A.2.1. If*

$f(s_1, \ldots, s_n)$ is a compound χ-term, then

$$canon_\chi(f(s_1, \ldots, s_n) \equiv canon_\chi(f(canon_\chi(s_1), \ldots, canon_\chi(s_n))).$$

Proof. By property 1 and 2 of *canon*, $\mathcal{T}_\chi \models canon(s_i) = s_i$ for each s_i. It follows that $\mathcal{T}_\chi \models f(canon(s_1), \ldots, canon(s_n)) = f(s_1, \ldots, s_n)$ by substitution. Thus, by property 1 of *canon*, $canon(f(canon(s_1), \ldots, canon(s_n))) \equiv canon(f(s_1, \ldots, s_n)$. The generalization to $canon_\chi$ is straightforward. □

Lemma A.4. *Suppose $\mathcal{T} \cup W \models \exists \overline{x}.\, X$, where $\overline{x} = free(X) - free(W)$ and $\mathcal{T} \cup X \models \exists \overline{y}.\, Y$, where $\overline{y} = free(Y) - free(X)$. Suppose also that $W \subseteq X$ and $X \subseteq Y$. Then $\mathcal{T} \cup W \models \exists \overline{w}.\, Y$, where $\overline{w} = \overline{x} \cup \overline{y} = free(Y) - free(W)$.*

Proof. Suppose $M \models_\rho W$. Then, because $\mathcal{T} \cup W \models \exists \overline{x}.\, X$, it follows that there exists a variable assignment ρ^* which differs from ρ only on $\overline{x}$ such that $M \models_{\rho^*} X$. Similarly, it follows from $\mathcal{T} \cup X \models \exists \overline{y}.\, Y$ that there exists a variable assignment ρ^{**} which differs from ρ^* only on $\overline{y}$ such that $M \models_{\rho^{**}} Y$. Thus, $M \models_\rho \exists \overline{x}, \overline{y}.\, Y$. But $\overline{x} = free(X) - free(W)$, $\overline{y} = free(Y) - free(X)$, and $W \subseteq X$ and $X \subseteq Y$, so it follows that $\overline{x} \cup \overline{y} = free(Y) - free(W)$. □

Lemma A.5. *If all global properties hold at the beginning of a procedure, and $P_=(all)$ is true at some line of the procedure, then G and $P_{34}[all](i)$ (for all i) also hold at that line.*

Proof. The global properties depend only on global state, so if the global state is unchanged, then clearly the global properties still hold. We now consider the properties in $P_{34}[all](i)$.

$\underline{P_=(\Phi - \mathcal{B}_i, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, \Lambda_*, *.find)}$**:** By $P_=(all)$.

$\underline{P_\subseteq(\mathcal{Q}, \mathcal{B}_i, \mathcal{V}, *.notify)}$**:** By $P_=(all)$.

$\underline{\mathcal{T}_i \cup \gamma_i(\Phi'_i \models \exists \overline{w}.\, \mathcal{B}_i)}$**:** By $\mathcal{B}'_i = \mathcal{B}_i$ and $\mathcal{B}'_i \subseteq \Phi'_i$. Note also that $\overline{w} = free(\mathcal{B}_i) - free(\Phi'_i)$, so $\overline{w} = \emptyset$, and thus, clearly, $\overline{w} \cap \mathcal{V}' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$.

$\underline{\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}'))}$: $\mathcal{Q} - \mathcal{Q}' = \emptyset$.

$\underline{\mathit{free}(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}}$: $\mathcal{Q} - \mathcal{Q}' = \emptyset$.

□

Lemma A.6. *If all global properties hold at the beginning of a procedure, and the property $P_{34}[all](i)$ holds at the end of a procedure, then all global properties except for G_9, G_{12} and G_{17} are automatically satisfied.*

Proof. Recall that property $P_{34}[all](i)$ implies that $P_{=}(\Phi - \mathcal{B}_i, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, \Lambda_*, *.\mathit{find})$, $P_{\subseteq}(\mathcal{Q}, \mathcal{B}_i, \mathcal{V}, *.\mathit{notify})$, $\mathcal{T}_i \cup \gamma_i(\Phi_i' \models \exists \overline{w}.\, \mathcal{B}_i) \wedge \overline{w} \cap \mathcal{V}\,' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$, where $\overline{w} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i')$, and $\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}')) \wedge \mathit{free}(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}$. We now show that the global properties hold.

$\underline{G_1}$: By $P_{=}(\mathcal{H}, \mathcal{I})$.

$\underline{G_2}$: By $\Phi = \Phi' \cup \mathcal{B}_i$ and $P_{=}(*.\mathit{find})$.

$\underline{G_3}$: By G_3 at the start of the procedure and $P_{=}(\mathcal{H})$, we have $\mathcal{T} \cup \mathcal{H} \models \exists \overline{w}.\, \Phi'$, where $\overline{w} = \mathit{free}(\Phi') - \mathit{free}(\mathcal{H})$. Then, by $P_{34}[all](i)$, we have $\gamma_i(\mathcal{T}_i \cup \Phi_i' \models \exists \overline{x}.\, \mathcal{B}_i)$, where $\overline{x} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i')$. It follows that $\mathcal{T} \cup \Phi_i' \models \exists \overline{x}.\, \mathcal{B}_i$, and thus $\mathcal{T} \cup \Phi' \models \exists \overline{x}.\mathcal{B}_i$, so $\mathcal{T} \cup \mathcal{H} \models \exists \overline{w}, \overline{x}.(\Phi' \cup \mathcal{B}_i)$. But $\Phi = \Phi' \cup \mathcal{B}_i$ (by $P_{=}(\Phi - \mathcal{B}_i)$ and $P_{\subseteq}(\mathcal{B}_i)$), so $\mathcal{T} \cup \mathcal{H} \models \exists \overline{w}, \overline{x}.\Phi$. It remains to show that $\overline{w} \cup \overline{x} = \mathit{free}(\Phi) - \mathit{free}(\mathcal{H})$. We know that $\mathit{free}(\Phi) = \mathit{free}(\Phi') \cup \mathit{free}(\mathcal{B}_i)$ and $\overline{w} = \mathit{free}(\Phi') - \mathit{free}(\mathcal{H})$. Now, $\overline{x} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i')$, but we also know that $\overline{x} \cap \mathcal{V}\,' = \emptyset$ and $\mathit{free}(\mathcal{H} \cup \Phi') \subseteq \mathcal{V}\,'$ (by G_8 at the start of the procedure), so $\overline{x} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\mathcal{H} \cup \Phi')$, and thus $\overline{w} \cup \overline{x} = \mathit{free}(\Phi) - \mathit{free}(\mathcal{H})$.

$\underline{G_4}$: By G_4 at the start of the procedure and $P_{=}(\mathcal{H}, \mathcal{N})$ and $\Phi = \Phi' \cup \mathcal{B}_i$, we have $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \mathcal{Q}' \cup \mathcal{N}$. It remains to show that $\mathcal{T} \cup \mathcal{H} \cup \Phi \models (\mathcal{Q} - \mathcal{Q}')$. But this follows from $\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}'))$.

$\underline{G_5}$: By $P_{=}(*.\mathit{find})$.

$\underline{G_6}$: By $P_{=}(*.\mathit{find})$.

$\underline{G_7}$: By $P_{=}(*.\mathit{find})$.

$\underline{G_8}$: By $P_{=}(\mathcal{N}, \mathcal{H}, \Lambda_*, *.\mathit{find})$ and $P_{\subseteq}(\mathcal{V})$, $\mathit{free}(\mathcal{N} \cup \mathcal{H} \cup \mathcal{F} \cup \Lambda) \subseteq \mathcal{V}$. It remains to show that $\mathit{free}((\mathcal{Q} - \mathcal{Q}') \cup (\Phi - \Phi')) \subseteq \mathcal{V}$. But this follows easily from $P_{34}[\mathit{all}](i)$.

$\underline{G_{11}}$: By $P_{=}(\mathcal{A}, *.\mathit{find})$.

$\underline{G_{13}}$: By $P_{=}(\mathcal{S})$.

$\underline{G_{14}}$: By $P_{=}(\mathcal{S}, *.\mathit{find})$.

$\underline{G_{15}}$: By $P_{=}(\mathcal{S}, *.\mathit{find})$.

$\underline{G_{16}}$: By definition, $P_{=}(G_{16}^{ok})$. G_{16} then follows by $P_{=}(*.\mathit{find})$ and $P_{\subseteq}(*.\mathit{notify})$.

$\underline{G_{18}}$: By $P_{=}(*.\mathit{find})$.

$\underline{G_{19}}$: By $P_{=}(\mathcal{S}, *.\mathit{find})$.

$\underline{G_{20}}$: By $P_{=}(\mathcal{A})$.

□

A.5.2 AddFact

Line 2:

0. G, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $P_2(e)$, $P_3[\Phi, \mathcal{H}](e)$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$
1. `AddFact(`e`)` [$\mathcal{H}$:= $\mathcal{H} \cup \{e\}$; $\mathcal{V}$:= $\mathcal{V} \cup \mathit{free}(e)$;]
2. $\underline{G}$, $\underline{P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e)}$, $e \in \mathcal{H}$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$

$\underline{G}$: Only those global properties which depend on $\mathcal{H}$ or $\mathcal{V}$ need be considered:

G_1: By G_1 at 0, $\mathcal{I}_0 \to \mathcal{T} \cup \mathcal{H}_0 \models \mathit{false}$. Since $\mathcal{I}$ is unchanged and $\mathcal{H}_0 \subseteq \mathcal{H}$, it follows that $\mathcal{I} \to \mathcal{T} \cup \mathcal{H} \models \mathit{false}$.

G_3: By G_3 at 0, $\mathcal{T} \cup \mathcal{H}_0 \models \exists \overline{w}.\, \Phi$, where $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\mathcal{H}_0)$. Since $\mathcal{H} = \mathcal{H}_0 \cup \{e\}$, it follows that $\mathcal{T} \cup \mathcal{H} \models \exists \overline{w}.\, \Phi$. But $P_3[\Phi, \mathcal{H}](e)$ at 0 ensures that $\mathit{free}(e) \cap \mathit{free}(\Phi - \mathcal{H}_0) = \emptyset$, so $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\mathcal{H})$.

G_4: By G_4 at 0, $\mathcal{T} \cup \mathcal{H}_0 \cup \Phi \models \mathcal{Q} \cup \mathcal{N}$. But since $\mathcal{H}_0 \subseteq \mathcal{H}$, it follows trivially that $\mathcal{T} \cup \mathcal{H}_0 \cup \Phi \models \mathcal{Q} \cup \mathcal{N}$.

G_8: By G_8 at 0, $\mathit{free}(\mathcal{Q} \cup \mathcal{N} \cup \Phi \cup \mathcal{H}_0 \cup \mathcal{F} \cup \Lambda) \subseteq \mathcal{V}_0$. But $\mathcal{H} = \mathcal{H}_0 \cup \{e\}$ and $\mathcal{V} = \mathcal{V}_0 \cup \mathit{free}(e)$, so it follows that $\mathit{free}(\mathcal{Q} \cup \mathcal{N} \cup \Phi \cup \mathcal{H} \cup \mathcal{F} \cup \Lambda) \subseteq \mathcal{V}$.

$\underline{P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e)}$: By $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$ at 0, $\neg\mathcal{I} \rightarrow (\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H})$. Then, since $\mathcal{H} = \mathcal{H}_0 \cup \{e\}$, it follows that $\neg\mathcal{I} \rightarrow (\mathcal{T} \cup \mathcal{N} \cup \Phi \cup \{e\} \models \mathcal{H})$.

Line 4:

```
2.   G, P4[Φ,H,I,N](e), e ∈ H, S2[I,S,*.find], S3[A,I,*.find]
3.   Q := {e};
4.   G, P4[Φ,H,I,N](e), Q = {e}, S2[I,S,*.find], S3[A,I,*.find]
```

$\underline{G}$: Only those global properties which depend on $\mathcal{Q}$ need be considered:

G_4: By G_4 at 2, $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \mathcal{Q}_0 \cup \mathcal{N}$. Then, since $e \in \mathcal{H}$ and $\mathcal{Q} = \{e\}$, it follows that $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \mathcal{Q} \cup \mathcal{N}$.

G_8: By G_8 at 2, $\mathit{free}(\mathcal{Q}_0 \cup \mathcal{N} \cup \Phi \cup \mathcal{H} \cup \mathcal{F} \cup \Lambda) \subseteq \mathcal{V}$, but again, $e \in \mathcal{H}$ and $\mathcal{Q} = \{e\}$, so $\mathcal{Q} \subseteq \mathcal{H}$, and thus $\mathit{free}(\mathcal{Q}) \subseteq \mathcal{V}$.

Line 6:

```
4.   G, P4[Φ,H,I,N](e), Q = {e}, S2[I,S,*.find], S3[A,I,*.find]
5.   REPEAT
6.     G, P5[Φ,H,I,N,Q](e), S2[I,S,*.find], S3[A,I,*.find]
...
26.    G, P1[Φ,H,I,N], P8[Φ,I,N,Q,Λ*,*.find](N+1),
       P9[Φ,I,N,Q,Λ*,*.find](i), S2[I,S,*.find], S3[A,I,*.find]
27.  UNTIL Q = ∅ OR I;
```

$\underline{P_5[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{Q}](e)}$: We must show $(P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e) \wedge \mathcal{Q} = \{e\}) \vee (P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}] \wedge \mathcal{Q} \neq \emptyset)$. From 4, this is clear since $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e) \wedge \mathcal{Q} = \{e\}$ at 4. From 26 it is also trivial since $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$ holds at 26 and $\mathcal{Q} \neq \emptyset$ by the loop condition.

Line 8:

```
 6.    G, P5[Φ,H,I,N,Q](e), S2[I,S,*.find], S3[A,I,*.find]
 7.    WHILE Q ≠ ∅ AND ¬I DO BEGIN
 8.      G, Q ≠ ∅, ¬I, P5[Φ,H,I,N,Q](e), S2[I,S,*.find], S3[A,I,*.find]
...
14.      G, P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find]
15.    END
```

$\underline{P_5[\Phi,\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{Q}](e)}$**:** We must show $(P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e) \wedge \mathcal{Q} = \{e\}) \vee (P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}] \wedge \mathcal{Q} \neq \emptyset)$. From 6, it follows trivially. From 14, it follows since $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ holds at 14 and $\mathcal{Q} \neq \emptyset$ by the loop condition.

Line 10:

```
 8.      G, Q ≠ ∅, ¬I, P5[Φ,H,I,N,Q](e), S2[I,S,*.find], S3[A,I,*.find]
 9.      Choose e* ∈ Q;
10.      G, P4[Φ,H,I,N](e*), Q ≠ ∅, ¬I, e* ∈ Q,
         P6[Φ,H](e*), free(e*) ⊆ V, S2[I,S,*.find], S3[A,I,*.find]
```

$\underline{P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)}$**:** By $P_5[\Phi,\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{Q}](e)$ at 8, either $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e) \wedge \mathcal{Q} = \{e\}$ or $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}] \wedge \mathcal{Q} \neq \emptyset$. Consider the first case: $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e) \wedge \mathcal{Q} = \{e\}$. Since $\mathcal{Q} = \{e\}$ and $e^* \in \mathcal{Q}$, it follows that $e \equiv e^*$. Thus, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$ follows from $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)$. In the other case, we have $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}] \wedge \neg\mathcal{Q} = \emptyset$. But $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ implies $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$.

$\underline{P_6[\Phi,\mathcal{H}](e^*)}$**:** By G_4 at 8, $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \mathcal{Q}$. But $e^* \in \mathcal{Q}$, so it follows that $\mathcal{T} \cup \mathcal{H} \cup \Phi \models e^*$.

$\underline{\mathit{free}(e^*) \subseteq \mathcal{V}}$**:** Follows by G_8 at 8 since $e^* \in \mathcal{Q}$.

Line 12:

```
10.      G, P4[Φ,H,I,N](e*), Q ≠ ∅, ¬I, e* ∈ Q,
         P6[Φ,H](e*), free(e*) ⊆ V, S2[I,S,*.find], S3[A,I,*.find]
11.      Q := Q − {e*};
12.      G, P4[Φ,H,I,N](e*), ¬I, P6[Φ,H](e*), free(e*) ⊆ V,
         S2[I,S,*.find], S3[A,I,*.find]
```

$\underline{G}$: Only those global properties which depend on $\mathcal{Q}$ need be considered. These are G_4 and G_8, and they follow from G_4 and G_8 respectively at 10 and the fact that $\mathcal{Q} \subset \mathcal{Q}_{10}$.

Line 14:

```
12.      G, P4[Φ,H,I,N](e*), ¬I, P6[Φ,H](e*), free(e*) ⊆ V,
         S2[I,S,*.find], S3[A,I,*.find]
13.      Assert(e*);
14.      G, P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find]
...
31. G, P4[Φ,H,I,N](e), ¬I, P6[Φ,H](e), free(e) ⊆ V, S2[I,S,*.find],
    S3[A,I,*.find]
32. Assert(e)
...
56. END Assert
57. G, P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find]
```

We must verify that the preconditions of `Assert` are met and that the postconditions of `Assert` imply the conditions at line 14. However, it is not hard to see that the preconditions of `Assert` match the properties at line 12 and that the postconditions of `Assert` are equivalent to the properties at 14.

Line 16:

```
 6.     G, P5[Φ,H,I,N,Q](e), S2[I,S,*.find], S3[A,I,*.find]
 7.     WHILE Q ≠ ∅ AND ¬I DO BEGIN
...
14.       G, P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find]
15.     END
16.     G, P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find]
```

$\underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]}$: From 6: By the loop condition, $\mathcal{Q} = \emptyset$ or $\mathcal{I}$. If $\mathcal{Q} = \emptyset$, then $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ follows from $P_5[\Phi,\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{Q}](e)$ at 6. If $\mathcal{I}$, then $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ is trivially true. From 14: trivial.

Line 18:

16. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$
17. `FOR` i `:= 1 TO` N `DO BEGIN`
18. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i)},\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],$ $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$

...

24. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),$ $P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$
25. `END`

$\underline{P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i)}$**:** We must show that $(\mathcal{Q}=\emptyset \wedge \neg\mathcal{I} \wedge \mathit{convex}) \rightarrow (\forall\, j, 0 < j < i.\ P_7[\Phi,\Lambda_*,*.\mathit{find}](j))$. From line 16, this is trivial since $i = 1$. For the transition from line 24, note that $i = i_{24}+1$. $P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i)$ follows easily by $P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i_{24})$ and $P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i_{24})$.

Line 22:

20. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \mathcal{Q}=\emptyset,\ \neg\mathcal{I},\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),\ \mathit{convex},$ $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$
21. `TheoryCheckSat`$_i$`();`
22. $\underline{G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),}$ $\underline{P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]}$

...

258. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \mathcal{Q}=\emptyset,\ \neg\mathcal{I},\ \mathit{convex},\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$
259. `TheoryCheckSat`$_i$`()`
260. $G,\ P_{=}(\mathit{all}-\{\mathcal{I},\mathcal{N},\mathcal{Q}\}),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),$ $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$

The conditions at 20 and 22 match the preconditions and postconditions of the theory-specific procedure `TheoryCheckSat`. The only exception is $P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i)$ at 22 which follows from $P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i)$ at 20 and the postcondition $P_{=}(\mathit{all}-\{\mathcal{I},\mathcal{N},\mathcal{Q}\})$ of `TheoryCheckSat`.

Line 26:

16. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$
17. `FOR` i `:= 1 TO` N `DO BEGIN`

...

24. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),$
$P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$
25. `END`
26. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](N+1)},$
$P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$

We assume there is at least one theory, so there is no possible transition from 16.

$\underline{P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](N+1)}$**:** This follows easily from $P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i)$ and $P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i)$ at 24 and the loop termination condition: $i = N$.

Line 28:

26. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](N+1),$
$P_9[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](i),\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$
27. `UNTIL` $\mathcal{Q} = \emptyset$ `OR` $\mathcal{I}$`;`
28. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \underline{P_{10}[\Phi,\mathcal{I},\mathcal{N},\Lambda_*,*.\mathit{find}]},\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$

$\underline{P_{10}[\Phi,\mathcal{I},\mathcal{N},\Lambda_*,*.\mathit{find}]}$**:** By $P_8[\Phi,\mathcal{I},\mathcal{N},\mathcal{Q},\Lambda_*,*.\mathit{find}](N+1)$ at 26, $(\mathcal{Q} = \emptyset \wedge \neg\mathcal{I} \wedge \mathit{convex}) \rightarrow (\forall\, j, 0 < j \leq N.\, P_7[\Phi,\Lambda_*,*.\mathit{find}](j))$. Then, by the loop termination condition, we have: $\mathcal{Q} = \emptyset \vee \mathcal{I}$. Suppose $\mathcal{Q} = \emptyset$, then we have $(\neg\mathcal{I} \wedge \mathit{convex}) \rightarrow (\forall\, j, 0 < j \leq N.\, P_7[\Phi,\Lambda_*,*.\mathit{find}](j))$, which is exactly $P_{10}[\Phi,\mathcal{I},\mathcal{N},\Lambda_*,*.\mathit{find}]$. If, on the other hand, $\mathcal{I}$ is `TRUE`, then $P_{10}[\Phi,\mathcal{I},\mathcal{N},\Lambda_*,*.\mathit{find}]$ follows trivially.

A.5.3 Assert

Line 35:

```
33.   G, P4[Φ,H,I,N](e), ¬I, P6[Φ,H](e), free(e) ⊆ V, S2[I,S,*.find],
      S3[A,I,*.find]
34.   e* := Simplify(e);
35.   G, P4[Φ,H,I,N](e*), ¬I, P6[Φ,H](e*), free(e*) ⊆ V, T ∪ Φ |= e ≃ e*,
      fr(e*), S2[I,S,*.find], S3[A,I,*.find], S4[S,*.find](e*), S5(e*)
...
147. G, free(e) ⊆ V
148. Simplify(e)
...
164. END Simplify
165. G, P=(A,H,I,N,S,*.find), P⊆(Φ,V), free(retval) ⊆ V, fr(retval),
     T ∪ Φ |= e' ≃ retval, P17[Φ](Φ'), S4[S,*.find](retval), S5(retval)
```

The preconditions of `Simplify` follow easily from the properties at 33. Thus, we simply must show that the postconditions imply the properties at 35.

$\underline{G}$**:** Follows by G at 165.

$\underline{\mathcal{T} \cup \Phi \models e \simeq e^*}$**:** By $\mathcal{T} \cup \Phi \models e' \simeq \mathit{retval}$ at 165.

$\underline{P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e^*)}$**:** To show that $\neg\mathcal{I} \rightarrow (\mathcal{T} \cup \mathcal{N} \cup \Phi \cup \{e^*\} \models \mathcal{H})$, suppose $\neg\mathcal{I}$ and $M \models_\rho \mathcal{T} \cup \mathcal{N} \cup \Phi \cup \{e^*\}$. We must show that $M \models_\rho \mathcal{H}$. First notice that by $\mathcal{T} \cup \Phi \models e \simeq e^*$ at 35 (which we just showed), it follows that $M \models_\rho \mathcal{T} \cup \mathcal{N} \cup \Phi \cup \{e\}$. Now, by $P_=(\mathcal{I}, \mathcal{N})$ and $P_\subseteq(\Phi)$ at 165, we have $\neg\mathcal{I}_{33}$ and $M \models_\rho \mathcal{T} \cup \mathcal{N}_{33} \cup \Phi_{33} \cup \{e\}$, so it follows from $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e)$ at 33 that $M \models_\rho \mathcal{H}_{33}$. But by $P_=(\mathcal{H})$ at 165, $\mathcal{H}_{33} = \mathcal{H}$, so $M \models_\rho \mathcal{H}$.

$\underline{\neg\mathcal{I}}$**:** By $\neg\mathcal{I}$ at 33 and $P_=(\mathcal{I})$ at 165.

$\underline{P_6[\Phi, \mathcal{H}](e^*)}$**:** To show that $\mathcal{T} \cup \mathcal{H} \cup \Phi \models e^*$, suppose $M \models_\rho \mathcal{T} \cup \mathcal{H} \cup \Phi$. We must show that $M \models_\rho e^*$. Now, by $P_=(\mathcal{H})$ and $P_\subseteq(\Phi)$ at 165, we have $M \models_\rho \mathcal{T} \cup \mathcal{H}_{33} \cup \Phi_{33}$. Then, by $P_6[\Phi, \mathcal{H}](e)$ at 33, $M \models_\rho e$. Finally, by $\mathcal{T} \cup \Phi \models e \simeq e^*$ at 35, it follows that $M \models_\rho e^*$.

$\underline{\mathit{free}(e^*) \subseteq \mathcal{V}}$**:** By $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$ at 165.

$\underline{\mathcal{T} \cup \Phi \models e \simeq e^*}$: By $\mathcal{T} \cup \Phi \models e' \simeq \mathit{retval}$ at 165.

$\underline{\mathit{fr}(e^*)}$: By $\mathit{fr}(\mathit{retval})$ at 165.

$\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$: By $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$ at 33, $P_=(\mathcal{I}, \mathcal{S}, *.\mathit{find})$ at 165.

$\underline{S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]}$: By $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$ at 33, $P_=(\mathcal{A}, \mathcal{I}, *.\mathit{find})$ at 165.

$\underline{S_4[\mathcal{S}, *.\mathit{find}](e^*)}$: By $S_4[\mathcal{S}, *.\mathit{find}](\mathit{retval})$ at 165.

$\underline{S_5(e^*)}$: By $S_5(\mathit{retval})$ at 165.

Line 39:

37. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi, \mathcal{H}](e^*)$, $\mathit{free}(e^*) \subseteq \mathcal{V}$, $\mathit{fr}(e^*)$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$
38. $\mathcal{N}$:= $\mathcal{N} \cup \{e^*\}$;
39. $\underline{G}$, $\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$

$\underline{G_4}$: By G_4 at 37, $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \mathcal{Q} \cup \mathcal{N}_{37}$. But $\mathcal{T} \cup \mathcal{H} \cup \Phi \models e^*$ by $P_6[\Phi, \mathcal{H}](e^*)$ at 37, so it follows that $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \mathcal{Q} \cup \mathcal{N}$.

$\underline{G_8}$: Follows from G_8 at 37 and $\mathit{free}(e^*) \subseteq \mathcal{V}$ at 37.

$\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$: By $\neg\mathcal{I}$ and $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e^*)$ at 37, $\mathcal{T} \cup \mathcal{N}_{37} \cup \Phi \cup \{e^*\} \models \mathcal{H}$. Thus, $\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H}$, from which $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$ follows easily.

Line 41:

35. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi, \mathcal{H}](e^*)$, $\mathit{free}(e^*) \subseteq \mathcal{V}$, $\mathcal{T} \cup \Phi \models e \simeq e^*$, $\mathit{fr}(e^*)$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_4[\mathcal{S}, *.\mathit{find}](e^*)$, $S_5(e^*)$
36. IF e^* is not a literal THEN BEGIN
...
40. END ELSE IF $\mathit{Op}(e^*) =$ '=' THEN BEGIN
41. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi, \mathcal{H}](e^*)$, $\mathit{free}(e^*) \subseteq \mathcal{V}$, $\mathit{fr}(e^*)$, $\underline{P_{11}(e^*)}$, $\mathit{Op}(e^*) =$ '=', $\underline{e^*[1] \not\equiv e^*[2]}$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_4[\mathcal{S}, *.\mathit{find}](e^*)$

$\underline{P_{11}(e^*)}$: By the if-condition at 36, e^* is a literal.

$\underline{e^*[1] \not\equiv e^*[2]}$: By $S_5(e^*)$ at 35, we have $\mathit{Op}(e^*) =$ '=' $\rightarrow e^*[1] \not\equiv e^*[2]$. Then, since $\mathit{Op}(e^*) =$ '=' at 41 (by the if-condition), $e^*[1] \not\equiv e^*[2]$.

Line 43:

41. $G,\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*),\ \neg\mathcal{I},\ P_6[\Phi,\mathcal{H}](e^*),\ \mathit{free}(e^*)\subseteq\mathcal{V},\ \mathit{fr}(e^*),\ P_{11}(e^*),$
$\mathit{Op}(e^*)=\text{`}=\text{'},\ e^*[1]\not\equiv e^*[2],\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ S_4[\mathcal{S},*.\mathit{find}](e^*)$
42. $\mathcal{E}$ `:= TheorySolve(`e^*`);`
43. $G,\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E}),\ \neg\mathcal{I},\ \mathit{free}(\mathcal{E})\subseteq\mathcal{V},\ P_{12}[\Phi,\mathcal{H}](\mathcal{E}),\ P_{14}[*.\mathit{find}](\mathcal{E}),$
$S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ S_7[\mathcal{S},*.\mathit{find}](\mathcal{E})$

...

267. $G,\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e),\ \neg\mathcal{I},\ P_6[\Phi,\mathcal{H}](e),\ \mathit{free}(e)\subseteq\mathcal{V},\ \mathit{fr}(e),\ P_{11}(e),$
$\mathit{Op}(e)=\text{`}=\text{'},\ e[1]\not\equiv e[2],\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ S_4[\mathcal{S},*.\mathit{find}](e)$
268. `TheorySolve(`e`)`
269. $G,\ P_=(\mathcal{H},\mathcal{I},\mathcal{N},*.\mathit{find}),\ P_\subseteq(\Phi),\ \mathit{free}(\mathit{retval})\subseteq\mathcal{V},\ P_{14}[*.\mathit{find}](\mathit{retval}),$
$P_{17}[\Phi](\Phi'),\ P_{35}[\Phi](e,\mathit{retval}),\ S_2[\mathcal{I},\mathcal{S},*.\mathit{find}],\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ S_7[\mathcal{S},*.\mathit{find}](\mathit{retval})$

The preconditions for `TheorySolve` match exactly the properties at line 41. We now show that the postconditions imply the properties at line 43.

$\underline{G}$**:** By G at 269.

$\underline{\neg\mathcal{I}}$**:** By $\neg\mathcal{I}$ at 41 and $P_=(\mathcal{I})$ at 269.

$\underline{P_{12}[\Phi,\mathcal{H}](\mathcal{E})}$**:** We must show that $\mathcal{T}\cup\mathcal{H}\cup\Phi\models\exists\overline{w}.\mathcal{E}$, where $\overline{w}=\mathit{free}(\mathcal{E})-\mathit{free}(\mathcal{H}\cup\Phi)$. First note that by $P_6[\Phi,\mathcal{H}](e^*)$ at 41, $\mathcal{T}\cup\mathcal{H}_{41}\cup\Phi_{41}\models e^*$. Then, by $P_=(\mathcal{H})$ and $P_\subseteq(\Phi)$ at 269, it follows that $\mathcal{T}\cup\mathcal{H}\cup\Phi\models e^*$. Now, by $P_{35}[\Phi](e,\mathit{retval})$ at 269, we have $\mathcal{T}\cup\Phi\models e^*\leftrightarrow\exists\overline{x}.\,\mathcal{E}$, where $\overline{x}=\mathit{free}(\mathcal{E})-\mathit{free}(e^*)$ and $\overline{x}\cap(\mathcal{V}_{41}\cup\mathit{free}(\Phi))=\emptyset$. It follows that $\mathcal{T}\cup\mathcal{H}\cup\Phi\models\exists\overline{x}.\,\mathcal{E}$. Now, note that $\overline{x}=\mathit{free}(\mathcal{E})-\mathit{free}(e^*)=\mathit{free}(\mathcal{E})-\mathit{free}(e^*)-\mathit{free}(\mathcal{H}\cup\Phi)$ since $\mathit{free}(\mathcal{H})\subseteq\mathcal{V}_{41}$ by G_8 at 41 and $P_=(\mathcal{H})$ at 269. But then $\overline{x}\subseteq\overline{w}$, so it follows that $\mathcal{T}\cup\mathcal{H}\cup\Phi\models\exists\overline{w}.\mathcal{E}$.

$\underline{P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E})}$**:** To show that $\neg\mathcal{I}\rightarrow(\mathcal{T}\cup\mathcal{N}\cup\Phi\cup\mathcal{E}\models\mathcal{H})$, note that we have already shown $\neg\mathcal{I}$ and then suppose that $M\models_\rho\mathcal{T}\cup\mathcal{N}\cup\Phi\cup\mathcal{E}$. We must show that $M\models_\rho\mathcal{H}$. As above, we have by $P_{35}[\Phi](e,\mathit{retval})$ at 269 that $\mathcal{T}\cup\Phi\models e^*\leftrightarrow\exists\overline{x}.\,\mathcal{E}$, where $\overline{x}=\mathit{free}(\mathcal{E})-\mathit{free}(e^*)$. Since $M\models_\rho\mathcal{E}$, clearly $M\models_\rho\exists\overline{x}.\,\mathcal{E}$, so therefore $M\models_\rho e^*$. Then, by $P_=(\mathcal{N})$ and $P_\subseteq(\Phi)$ at 269, we can conclude that $M\models_\rho\mathcal{T}\cup\mathcal{N}_{41}\cup\Phi_{41}\cup\{e^*\}$. It then follows from $P_6[\Phi,\mathcal{H}](e^*)$ at 41 that $M\models_\rho\mathcal{H}_{41}$. Finally, by $P_=(\mathcal{H})$ at 269, we conclude that $\mathcal{M}\models_\rho\mathcal{H}$.

$\underline{\mathit{free}(\mathcal{E})\subseteq\mathcal{V}}$**:** By $\mathit{free}(\mathit{retval})\subseteq\mathcal{V}$ at 269.

$\underline{P_{14}[*.find](\mathcal{E})}$: By $P_{14}[*.find](retval)$ at 269.

$\underline{S_2[\mathcal{I}, \mathcal{S}, *.find]}$: By $S_2[\mathcal{I}, \mathcal{S}, *.find]$ at 269.

$\underline{S_3[\mathcal{A}, \mathcal{I}, *.find]}$: By $S_3[\mathcal{A}, \mathcal{I}, *.find]$ at 269.

$\underline{S_7[\mathcal{S}, *.find](\mathcal{E})}$: By $S_7[\mathcal{S}, *.find](retval)$ at 269.

Line 45:

43. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](\mathcal{E})$, $\neg\mathcal{I}$, $free(\mathcal{E}) \subseteq \mathcal{V}$, $P_{12}[\Phi, \mathcal{H}](\mathcal{E})$, $P_{14}[*.find](\mathcal{E})$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$, $S_7[\mathcal{S}, *.find](\mathcal{E})$
44. `AssertEqualities(`$\mathcal{E}$`);`
45. $\underline{G,\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ S_2[\mathcal{I}, \mathcal{S}, *.find],\ S_3[\mathcal{A}, \mathcal{I}, *.find]}$

...

58. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](\mathcal{E})$, $\neg\mathcal{I}$, $P_{12}[\Phi, \mathcal{H}](\mathcal{E})$, $P_{14}[*.find](\mathcal{E})$, $free(\mathcal{E}) \subseteq \mathcal{V}$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$, $\underline{S_{12}[\mathcal{S}, *.find](\mathcal{E})}$
59. `AssertEqualities(`$\mathcal{E}$`)`

...

97. `END AssertEqualities`
98. G, $P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$, $\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})$, $\mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R})$, $\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$

First, consider the preconditions of `AssertEqualities`. All preconditions except for $S_{12}[\mathcal{S}, *.find](\mathcal{E})$ are trivial.

$\underline{S_{12}[\mathcal{S}, *.find](\mathcal{E})}$: We must show that at 43, $false \in \mathcal{E} \vee ((S_6(\mathcal{E}) \vee S_{10}[\mathcal{S}, *.find](\mathcal{E})) \wedge S_{11}[\mathcal{S}, *.find](\mathcal{E}))$. If $false \in \mathcal{E}$, then $S_{12}[\mathcal{S}, *.find](\mathcal{E})$ follows trivially. Suppose $false \notin \mathcal{E}$. Then, by $S_7[\mathcal{S}, *.find](\mathcal{E})$ at 43, $S_4[\mathcal{S}, *.find](\mathcal{E}) \wedge S_6(\mathcal{E})$, and by $P_{14}[*.find](\mathcal{E})$ at 43, $\forall e \in \mathcal{E}.\, fr(e) \wedge P_{13}(\mathcal{E})$ $S_{12}[\mathcal{S}, *.find](\mathcal{E})$ will follow if we can show $S_{11}[\mathcal{S}, *.find](\mathcal{E})$. To this end, suppose $e \in \mathcal{E}$. We have $fr(e)$ by $\forall e \in \mathcal{E}.\, fr(e)$ and $S_4[\mathcal{S}, *.find](e)$ by $S_4[\mathcal{S}, *.find](\mathcal{E})$. Finally, we must show $e[2] \equiv canon_\chi(\mathcal{S}(e[2]))$. Suppose $hf(e[2])$. Then, since $fr(e)$, we must have $e[2].find \equiv e[2]$. It then follows from $\neg\mathcal{I}$ and $S_2[\mathcal{I}, \mathcal{S}, *.find]$ that $e[2] \equiv canon_\chi(\mathcal{S}(e[2]))$. If $\neg hf(e[2])$, then $e[2] \equiv canon_\chi(\mathcal{S}(e[2]))$ by $S_4[\mathcal{S}, *.find](e)$.

We now show that the postconditions imply the properties at 45.

$\underline{G}$: By G at 98.

$\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$: By $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$ at 98.

$\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$: Assume $\neg\mathcal{I}$. We must show $\forall e.\, (\mathit{hf}(e) \rightarrow \mathit{find}^*(e) \equiv \mathit{canon}_\chi(\mathcal{S}(e)))$. Suppose then that e is an expression $e \in \mathit{HF}$. Now consider two cases. If $e \in \mathit{HF}_{43}$, then $e \in \mathcal{R}_{43}$ by $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$ and $\neg\mathcal{I}$ at 43. Then, by $\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$ at 98 (and because we have assumed $\neg\mathcal{I}$), $e \in \mathcal{R}$, so $\mathit{find}^*(e) \equiv \mathit{canon}_\chi(\mathcal{S}(e))$. Suppose on the other hand that $e \notin \mathit{HF}_{43}$. Then since $e \in \mathit{HF}$, $e \in \mathcal{M}$ at 98. It follows by $\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})$ that $e \in \mathcal{R}$, which implies $\mathit{find}^*(e) \equiv \mathit{canon}_\chi(\mathcal{S}(e))$.

$\underline{S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]}$: By $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$ at 98.

Line 49:

47. G, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e^*)$, $\neg\mathcal{I}$, $P_6[\Phi, \mathcal{H}](e^*)$, $\mathcal{T} \cup \Phi \models e \simeq e^*$, $\mathit{fr}(e^*)$, $e^* \equiv \mathit{false}$, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$
48. `I := TRUE;`
49. $\underline{G}$, $\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$, $\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$, $\underline{S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]}$

$\underline{G}$: Only G_1 is affected. We must show that $\mathcal{T} \cup \mathcal{H}$ is unsatisfiable. Suppose $\mathcal{T} \cup \mathcal{H}$ is satisfiable. Then there exists a model M and interpretation ρ such that $M \models_\rho \mathcal{T} \cup \mathcal{H}$. Then, by G_3 at 47, we can construct ρ^* so that $M \models_{\rho^*} \mathcal{T} \cup \mathcal{H} \cup \Phi$. But, by $P_6[\Phi, \mathcal{H}](e^*)$ and $e^* \equiv \mathit{false}$ at 47, it then follows that $M \models_{\rho^*} \mathit{false}$, which is a contradiction. Thus, it must be the case that $\mathcal{T} \cup \mathcal{H}$ is unsatisfiable.

$\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$,$\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$,$\underline{S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]}$: These are trivial since $\mathcal{I}$ is `TRUE`.

Line 53:

```
51.   G, P4[Φ,H,I,N](e*), ¬I, P6[Φ,H](e*), free(e*) ⊆ V, fr(e*), P11(e*),
      Op(e*) ≠ '=', S2[I,S,*.find], S3[A,I,*.find], S4[S,*.find](e*)
52.   AssertFormula(e*);
53.   G, P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find]
...
 99. G, free(e) ⊆ V, fr(e), P11(e), P20[H](Φ,e), S4[S,*.find](e)
100. AssertFormula(e)
...
112. END AssertFormula
113. G, P=(H,S), P⊆(Φ,V,N,F,R), P17[Φ](Φ' ∪ {e}), P21[*.find],
      P25[*.find](e), e ∈ Φ, A = A' ∪ {e}, M ⊆ R
```

First consider the preconditions of `AssertFormula`.

$P_{20}[\mathcal{H}](\Phi, e^*)$**:** Let $W = \Phi$, $X = \emptyset$, $Y = \{e^*\}$, and $Z = \emptyset$. Clearly, $\{W, X, Z\}$ is a partition of Φ. $\mathcal{T} \cup \mathcal{H} \models \exists\, \overline{w}.\, \Phi$, where $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\mathcal{H})$ by G_3. Also, $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \exists \overline{x}, \overline{y}.\, (X \cup Y)$ simply reduces to $\mathcal{T} \cup \mathcal{H} \cup \Phi \models \exists \overline{y}.e^*$ which follows from $P_6[\Phi, \mathcal{H}](e^*)$ at 51.

Now we consider the properties at line 53.

G**:** By G at 113.

$P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$**:** Suppose $\neg\mathcal{I}$ and $M \models_\rho \mathcal{T} \cup \mathcal{N} \cup \Phi$. We must show that $M \models_\rho \mathcal{H}$. First note that by $P_\subseteq(\Phi, \mathcal{N})$ at 113, we have $M \models_\rho \mathcal{T} \cup \mathcal{N}_{51} \cup \Phi_{51}$. Also, by $e \in \Phi$ at 113, $e^* \in \Phi$, so $M \models_\rho e^*$, and thus $M \models_\rho \mathcal{T} \cup \mathcal{N}_{51} \cup \Phi_{51} \cup \{e^*\}$. Then, by $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](e^*)$ at 51, $M \models_\rho \mathcal{H}_{51}$. Finally, by $P_=(\mathcal{H})$ at 113, $M \models_\rho \mathcal{H}$.

$S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$**:** Assume $\neg\mathcal{I}$. We must show $\forall e.\, (\mathit{hf}(e) \rightarrow \mathit{find}^*(e) \equiv \mathit{canon}_\chi(\mathcal{S}(e)))$. Suppose then that e is an expression $e \in \mathit{HF}$. Now consider two cases. If $e \in \mathit{HF}_{51}$, then $e \in \mathcal{R}_{51}$ by $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$ and $\neg\mathcal{I}$ at 51. Then, by $P_\subseteq(\mathcal{R})$ at 113, $e \in \mathcal{R}$, so $\mathit{find}^*(e) \equiv \mathit{canon}_\chi(\mathcal{S}(e))$. Suppose on the other hand that $e \notin \mathit{HF}_{51}$. Then since $e \in \mathit{HF}$, $e \in \mathcal{M}$ at 113. It follows by $\mathcal{M} \subseteq \mathcal{R}$ that $e \in \mathcal{R}$, which implies $\mathit{find}^*(e) \equiv \mathit{canon}_\chi(\mathcal{S}(e))$.

$\underline{S_3[\mathcal{A},\mathcal{I},*.find]}$: Assume $\neg\mathcal{I}$. We must show $\forall e \in \mathcal{A}.\ (Op(e) = \text{`='} \rightarrow e[1] \sim e[2])$. Consider $d \in \mathcal{A}$. Suppose $d \in \mathcal{A}_{51}$. Then $Op(d) = \text{`='} \rightarrow d[1] \sim_{51} d[2]$. But by $P_{\subseteq}(\mathcal{F})$ at 113 it follows that $Op(d) = \text{`='} \rightarrow d[1] \sim d[2]$. Suppose on the other hand that $d \notin \mathcal{A}_{51}$. Then since $\mathcal{A} = \mathcal{A}' \cup \{e\}$ at 113, it follows that $\mathcal{A} = \mathcal{A}_{51} \cup \{e^*\}$ at 53, so it must be the case that $d \equiv e^*$. Since we know $Op(e^*) \neq \text{`='}$, it follows trivially that $Op(d) = \text{`='} \rightarrow d[1] \sim d[2]$.

Line 55:

35. $G,\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*),\ \neg\mathcal{I},\ P_6[\Phi,\mathcal{H}](e^*),\ free(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,$ $fr(e^*),\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find],\ S_4[\mathcal{S},*.find](e^*),\ S_5(e^*)$
36. `IF ...`
39. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
40. `END ELSE IF ...`
45. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
46. `END ELSE IF ...`
49. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
50. `END ELSE IF` $e^* \not\equiv$ *true* `THEN BEGIN`

...

53. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$
54. `END`
55. $G,\ \underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]},\ S_2[\mathcal{I},\mathcal{S},*.find],\ S_3[\mathcal{A},\mathcal{I},*.find]$

Note that the conditions at line 55 follow trivially from the conditions at the end of each if-block. Thus, the only case which is not obvious is when none of the if-conditions are true. In this case, $S_2[\mathcal{I},\mathcal{S},*.find]$ and $S_3[\mathcal{A},\mathcal{I},*.find]$ follow trivially from 35. To show $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, note that $e^* \equiv true$. Thus, by $\neg\mathcal{I}$ and $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e^*)$ at 35, $\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H}$, from which $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ follows easily.

A.5.4 AssertEqualities

Line 64:

62. $G,\ P_{=}(all),\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E}),\ P_{12}[\Phi,\mathcal{H}](\mathcal{E}),\ false \in \mathcal{E},\ P_{14}[*.find](\mathcal{E})$
63. $\mathcal{I}$ `:= TRUE;`
64. $\underline{G},\ P_{=}(all - \{\mathcal{I}\}),\ \underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]},\ \mathcal{I}$

$\underline{G}$: Only G_1 is affected. We must show that $\mathcal{T} \cup \mathcal{H}$ is unsatisfiable. Suppose $\mathcal{T} \cup \mathcal{H}$ is satisfiable. Then there exists a model M and interpretation ρ such that $M \models_\rho \mathcal{T} \cup \mathcal{H}$. Then, by G_3 at 62, we can construct ρ^* so that $M \models_{\rho^*} \mathcal{T} \cup \mathcal{H} \cup \Phi$. But, by $P_{12}[\Phi, \mathcal{H}](\mathcal{E})$ and $\mathit{false} \in \mathcal{E}$ at 62, it then follows that $M \models_{\rho^*} \mathit{false}$, which is a contradiction. Thus, it must be the case that $\mathcal{T} \cup \mathcal{H}$ is unsatisfiable.

$\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$: Trivial since $\mathcal{I}$ is `TRUE`.

Line 66:

60. G, $P_=(all)$, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](\mathcal{E})$, $\neg\mathcal{I}$, $\mathit{free}(\mathcal{E}) \subseteq \mathcal{V}$, $P_{12}[\Phi, \mathcal{H}](\mathcal{E})$, $P_{14}[*.\mathit{find}](\mathcal{E})$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $S_{12}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$
61. `IF` $\mathit{false} \in \mathcal{E}$ `THEN BEGIN`
...
65. `END ELSE BEGIN` [$\mathcal{A}^* :=$ `;` $\Phi^* :=$ `;` $\mathcal{N}^* :=$ `;` $X :=$ `;` $\emptyset$]
66. G, $P_=(all)$, $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](\mathcal{E})$, $\neg\mathcal{I}$, $\mathit{free}(\mathcal{E}) \subseteq \mathcal{V}$, $P_{12}[\Phi, \mathcal{H}](\mathcal{E})$, $\mathit{false} \notin \mathcal{E}$, $\underline{\forall e \in \mathcal{E}.\, \mathit{fr}(e)}$, $\underline{P_{13}(\mathcal{E})}$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\underline{S_6(\mathcal{E}) \vee S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})}$, $\underline{S_{11}[\mathcal{S}, *.\mathit{find}](\mathcal{E})}$

$\underline{\forall e \in \mathcal{E}.\, \mathit{fr}(e)}$: By $P_{14}[*.\mathit{find}](\mathcal{E})$ at 60 and $\mathit{false} \notin \mathcal{E}$ at 66.

$\underline{P_{13}(\mathcal{E})}$: By $P_{14}[*.\mathit{find}](\mathcal{E})$ at 60 and $\mathit{false} \notin \mathcal{E}$ at 66.

$\underline{S_6(\mathcal{E}) \vee S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})}$: By $S_{12}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ at 60 and $\mathit{false} \notin \mathcal{E}$ at 66.

$\underline{S_{11}[\mathcal{S}, *.\mathit{find}](\mathcal{E})}$: By $S_{12}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ at 60 and $\mathit{false} \notin \mathcal{E}$ at 66.

Line 68:

65. `END ELSE BEGIN` [$\mathcal{A}^* := \mathcal{A};\ \Phi^* := \Phi;\ \mathcal{N}^* := \mathcal{N};\ X := \emptyset$]
66. $G,\ P_{=}(all),\ P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\mathcal{E}),\ \neg\mathcal{I},\ \mathit{free}(\mathcal{E}) \subseteq \mathcal{V},\ P_{12}[\Phi,\mathcal{H}](\mathcal{E}),\ \mathit{false} \notin \mathcal{E},$ $\forall e \in \mathcal{E}.\, \mathit{fr}(e),\ P_{13}(\mathcal{E}),\ S_3[\mathcal{A},\mathcal{I},*.\mathit{find}],\ S_6(\mathcal{E}) \vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E}),$ $S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E})$
67. `FOREACH` $e \in \mathcal{E}$ `DO BEGIN`
68. $G,\ P_{=}(\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\mathcal{V},\mathcal{F},\mathcal{R}),\ \underline{P_{15}[\Phi,\mathcal{N}](\Phi^*,\mathcal{N}^*,X)},\ P_{13}(\mathcal{E}),\ X \subseteq \mathcal{E},$ $\underline{P_{16}[*.\mathit{find}](X,\mathcal{E})},\ \underline{P_{17}[\Phi](\Phi^* \cup X)},\ \mathit{free}(\mathcal{E}) \subseteq \mathcal{V}',\ S_6(\mathcal{E}) \vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E}),$ $S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E}),\ \underline{\mathcal{A} = \mathcal{A}^* \cup X},\ \underline{S_{13}[*.\mathit{find}](\mathcal{A}^*)},\ \underline{\mathcal{M} \subseteq \mathcal{R}},\ e \in \mathcal{E},\ e \notin X$

...

70. $G,\ P_{=}(\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\mathcal{V},\mathcal{F},\mathcal{R}),\ P_{15}[\Phi,\mathcal{N}](\Phi^*,\mathcal{N}^*,X),\ P_{13}(\mathcal{E}),\ X \subseteq \mathcal{E},$ $P_{16}[*.\mathit{find}](X,\mathcal{E}),\ P_{17}[\Phi](\Phi^* \cup X),\ \mathit{free}(\mathcal{E}) \subseteq \mathcal{V}',\ S_6(\mathcal{E}) \vee S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E}),$ $S_{11}[\mathcal{S},*.\mathit{find}](\mathcal{E}),\ \mathcal{A} = \mathcal{A}^* \cup X,\ S_{13}[*.\mathit{find}](\mathcal{A}^*),\ \mathcal{M} \subseteq \mathcal{R}$
71. `END`

The variables $\mathcal{A}^*$, Φ^*, $\mathcal{N}^*$, and X are helper variables which aid the proof. $\mathcal{A}^*$, Φ^*, and $\mathcal{N}^*$ simply store for future reference the value of their respective state variables at line 65. X is used to track which elements of $\mathcal{E}$ have been processed by the loop at lines 67 to 71. Since the properties at 68 and 70 are identical (except for the trivial ones which mention e), we only need to consider the transition from line 66 to 68.

$\underline{P_{15}[\Phi,\mathcal{N}](\Phi^*,\mathcal{N}^*,X)}$**:** Trivial since $\Phi^* = \Phi$, $\mathcal{N}^* = \mathcal{N}$, and $X = \emptyset$ at 66.

$\underline{P_{16}[*.\mathit{find}](X,\mathcal{E})}$**:** Since $X = \emptyset$ at 66, this follows trivially from $\forall e \in \mathcal{E}.\, \mathit{fr}(e)$ at at 66.

$\underline{P_{17}[\Phi](\Phi^* \cup X)}$**:** Trivial since $\Phi^* = \Phi$ and $X = \emptyset$ at 66.

$\underline{\mathcal{A} = \mathcal{A}^* \cup X}$**:** Trivial since $\mathcal{A} = \mathcal{A}^*$ and $X = \emptyset$ at 66.

$\underline{S_{13}[*.\mathit{find}](\mathcal{A}^*)}$**:** By $\neg\mathcal{I}$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, and $\mathcal{A} = \mathcal{A}^*$ at 66.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$**:** By $P_{=}(all)$ at 66, $\mathcal{M} = \emptyset$.

Line 70:

```
68.   G, P_=(H,S), P_⊆(V,F,R), P_15[Φ,N](Φ*,N*,X), P_13(E), X ⊆ E,
      P_16[*.find](X,E), P_17[Φ](Φ* ∪ X), free(E) ⊆ V', S_6(E) ∨ S_10[S,*.find](E),
      S_11[S,*.find](E), A = A* ∪ X, S_13[*.find](A*), M ⊆ R, e ∈ E, e ∉ X
69.   AssertFormula(e); [ X := X ∪ {e}; ]
70.   G, P_=(H,S), P_⊆(V,F,R), P_15[Φ,N](Φ*,N*,X), P_13(E), X ⊆ E,
      P_16[*.find](X,E), P_17[Φ](Φ* ∪ X), free(E) ⊆ V', S_6(E) ∨ S_10[S,*.find](E),
      S_11[S,*.find](E), A = A* ∪ X, S_13[*.find](A*), M ⊆ R
...
99.  G, free(e) ⊆ V, fr(e), P_11(e), P_20[H](Φ,e), S_4[S,*.find](e)
100. AssertFormula(e)
...
112. END AssertFormula
113. G, P_=(H,S), P_⊆(Φ,V,N,F,R), P_17[Φ](Φ' ∪ {e}), P_21[*.find],
     P_25[*.find](e), e ∈ Φ, A = A' ∪ {e}, M ⊆ R
```

First we consider the preconditions of `AssertFormula`.

$\underline{free(e) \subseteq \mathcal{V}}$**:** Follows from $free(\mathcal{E}) \subseteq \mathcal{V}'$, $P_{\subseteq}(\mathcal{V})$, and $e \in \mathcal{E}$ at 68.

$\underline{fr(e)}$**:** By $P_{16}[*.find](X, \mathcal{E})$ and $e \notin X$ at 68.

$\underline{P_{20}[\mathcal{H}](\Phi, e)}$**:** Let $W = \Phi^*$, $X = X - \Phi^*$, $Y = \mathcal{E} - X$, and $Z = \Phi - (\Phi^* \cup X)$. It is not hard to see that $\{\Phi^*, X - \Phi^*, \Phi - (\Phi^* \cup X)\}$ is a partition of Φ: the sets are clearly disjoint (by construction). Then, their union is Φ because $\Phi^* \subseteq \Phi$ and $X \subseteq \Phi$ (by $P_{15}[\Phi, \mathcal{N}](\Phi^*, \mathcal{N}^*, X)$ at 68). Next we must show that $\mathcal{T} \cup \mathcal{H} \models \exists \overline{w}.\, \Phi^*$, where $\overline{w} = free(\Phi^*) - free(\mathcal{H})$. But this follows easily from G_3 at 66 and $P_=(\mathcal{H})$ at 66 and 68. Next, we must show that $\mathcal{T} \cup \mathcal{H} \cup \Phi^* \models \exists \overline{x}, \overline{y}.((X - \Phi^*) \cup (\mathcal{E} - X))$ where $\overline{x} = free(X - \Phi^*) - free(\mathcal{H} \cup \Phi^*)$ and $\overline{y} = free(\mathcal{E} - X) - free(\mathcal{H} \cup \Phi^* \cup (X - \Phi^*))$. With some effort and the the fact that $X \subseteq \mathcal{E}$ at 68, it can be shown that this is equivalent to $\mathcal{T} \cup \mathcal{H} \cup \Phi^* \models \exists \overline{x}.\, (\mathcal{E} - (\Phi^* \cap X))$ where $\overline{x} = free(\mathcal{E} - (\Phi^* \cap X)) - free(\mathcal{H} \cup \Phi^*)$. This certainly follows from the stronger requirement $\mathcal{T} \cup \mathcal{H} \cup \Phi^* \models \exists \overline{x}.\, \mathcal{E}$ where $\overline{x} = free(\mathcal{E}) - free(\mathcal{H} \cup \Phi^*)$. But by $P_=(\mathcal{H})$ at 66 and 68, this follows from $P_{12}[\Phi, \mathcal{H}](\mathcal{E})$ at 66. Finally, we must show that $\mathcal{T} \cup \Phi^* \cup (X - \Phi^*) \models \exists \overline{z}.\, (\Phi - (\Phi^* \cup X))$ and $\overline{z} \cap free(\mathcal{H} \cup \{e\}) = \emptyset$,

where $\overline{z} = \mathit{free}(\Phi - (\Phi^* \cup X)) - \mathit{free}(\Phi^* \cup (X - \Phi^*))$. This follows from the stronger condition $\mathcal{T} \cup \Phi^* \cup X \models \exists \overline{z}.\ \Phi$ and $\overline{z} \cap \mathit{free}(\mathcal{H} \cup \{e\}) = \emptyset$, where $\overline{z} = \mathit{free}(\Phi) - \mathit{free}(\Phi^* \cup X)$ which follows from $P_{17}[\Phi](\Phi^* \cup X)$ at 68, together with $e \in \mathcal{E}$ and $\mathit{free}(\mathcal{E}) \subseteq \mathcal{V}'$ at 68, and $P_{=}(\mathcal{H})$ at 68 and G_8 at 58. The last condition, $e \in (\mathcal{E} - X)$ follows easily from $e \in \mathcal{E}$ and $e \notin X$ at 68.

$S_4[\mathcal{S}, *.\mathit{find}](e)$**:** By $S_{11}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ and $e \in \mathcal{E}$ at 68.

Now we consider the properties at line 70.

G**:** By G at 113.

$P_{=}(\mathcal{H}, \mathcal{S})$**:** By $P_{=}(\mathcal{H}, \mathcal{S})$ at 68 and 113.

$P_{\subseteq}(\mathcal{V}, \mathcal{F}, \mathcal{R})$**:** By $P_{\subseteq}(\mathcal{V}, \mathcal{F}, \mathcal{R})$ at 68 and 113.

$P_{15}[\Phi, \mathcal{N}](\Phi^*, \mathcal{N}^*, X)$**:** By $P_{15}[\Phi, \mathcal{N}](\Phi^*, \mathcal{N}^*, X)$ at 68 and $P_{\subseteq}(\Phi, \mathcal{N})$ at 113.

$P_{13}(\mathcal{E})$**:** By $P_{13}(\mathcal{E})$ at 68.

$X \subseteq \mathcal{E}$**:** By $X_{68} \subseteq \mathcal{E}$, $e \in \mathcal{E}$, and $X = X_{68} \cup \{e\}$.

$P_{16}[*.\mathit{find}](X, \mathcal{E})$**:** We must show $\forall e \in \mathcal{E}.\ [(e \in X \rightarrow (e[1].\mathit{find} \equiv e[1] \wedge e[2].\mathit{find} \equiv e[2])) \wedge (e \notin X \rightarrow \mathit{fr}(e))]$. Suppose $e \in \mathcal{E}$. Suppose further that $e \in X$. If $e \in X_{68}$, then $e[1].\mathit{find}_{68} \equiv e[1]$ and $e[2].\mathit{find}_{68} \equiv e[2]$, so by $P_{21}[*.\mathit{find}]$ at 113, $e[1].\mathit{find} \equiv e[1]$ and $e[2].\mathit{find} \equiv e[2]$. If $e \notin X_{68}$, then by the definition of X, e must have been the argument to the call to `AssertFormula`, so $e[1].\mathit{find} \equiv e[1]$ and $e[2].\mathit{find} \equiv e[2]$ by $P_{25}[*.\mathit{find}](e)$ at 113. Finally, suppose that $e \notin X$. It follows that $e \notin X_{68}$, so $\mathit{fr}(e)$ was true at 68. By $P_{21}[*.\mathit{find}]$ at 113, $\mathit{fr}(e)$ is true at 70 as well.

$P_{17}[\Phi](\Phi^* \cup X)$**:** We must show that $\mathcal{T} \cup \Phi^* \cup X \models \exists \overline{w}.\ \Phi$ and $\overline{w} \cap \mathcal{V}' = \emptyset$, where $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\Phi^* \cup X)$. Suppose that $M \models_\rho \mathcal{T} \cup \Phi^* \cup X$. Now, by $P_{17}[\Phi](\Phi^* \cup X)$ at 68, we have $\mathcal{T} \cup \Phi^* \cup X_{68} \models \exists \overline{x}.\ \Phi_{68}$ and $\overline{x} \cap \mathcal{V}' = \emptyset$, where $\overline{x} = \mathit{free}(\Phi_{68}) - \mathit{free}(\Phi^* \cup X_{68})$. It follows that $\mathcal{T} \cup \Phi^* \cup X \models \exists \overline{x}.\ \Phi_{68}$ since $X = X_{68} \cup \{e\}$. Further, $\overline{x} = \mathit{free}(\Phi_{68}) - \mathit{free}(\Phi^* \cup X)$ since $\overline{x} \cap \mathcal{V}' = \emptyset$ and

$\mathit{free}(e) \subseteq \mathcal{V}'$. Thus $M \models_\rho \exists \overline{x}.\, \Phi_{68}$, so there is a variable assignment ρ^* such that $M \models_{\rho^*} \mathcal{T} \cup \Phi^* \cup X \cup \Phi_{68}$ where ρ^* differs from ρ at most only on the values of $\overline{x}$. Now, by $P_{17}[\Phi](\Phi' \cup \{e\})$ at 113, we have $\mathcal{T} \cup \Phi_{68} \cup \{e\} \models \exists \overline{y}.\, \Phi$ and $\overline{y} \cap \mathcal{V}_{68} = \emptyset$, where $\overline{y} = \mathit{free}(\Phi) - \mathit{free}(\Phi_{68} \cup \{e\})$. It then follows that $M \models_{\rho^*} \exists \overline{y}.\, \Phi$. Thus, we can conclude that $M \models_\rho \exists \overline{x}, \overline{y}.\, \Phi$. Finally, we show that if $\overline{w} = \overline{x} \cup \overline{y}$, then $\overline{w} \cap \mathcal{V}' = \emptyset$, and $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\Phi^* \cup X)$. First note that we already have $\overline{x} \cap \mathcal{V}' = \emptyset$. It is easy to see that $\overline{y} \cap \mathcal{V}' = \emptyset$ since $\overline{y} \cap \mathcal{V}_{68} = \emptyset$ and $P_\subseteq(\mathcal{V})$ at 68. Now, let $W = \Phi^* \cup X$, then $\overline{w} = \overline{x} \cup \overline{y} = (\mathit{free}(\Phi_{68}) - \mathit{free}(W)) \cup (\mathit{free}(\Phi) - \mathit{free}(\Phi_{68} \cup \{e\}))$. But $\Phi_{68} \cup \{e\} = \Phi_{68} \cup W$, since $\Phi^* \subseteq \Phi_{68}$ and $X_{68} \subseteq \Phi_{68}$ by $P_{15}[\Phi, \mathcal{N}](\Phi^*, \mathcal{N}^*, X)$ at 68 (and $X = X_{68} \cup \{e\}$), so $\overline{w} = (\mathit{free}(\Phi_{68}) - \mathit{free}(W)) \cup (\mathit{free}(\Phi) - \mathit{free}(\Phi_{68} \cup W))$. Finally, since $\Phi_{68} \subseteq \Phi$ by $P_\subseteq(\Phi)$ at 113, it is not hard to see that $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(W) = \mathit{free}(\Phi) - \mathit{free}(\Phi^* \cup X)$.

$\underline{\mathit{free}(\mathcal{E}) \subseteq \mathcal{V}'}$**:** By $\mathit{free}(\mathcal{E}) \subseteq \mathcal{V}'$ at 68.

$\underline{S_6(\mathcal{E}) \vee S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})}$**:** Suppose $S_6(\mathcal{E})$ at 68. Then $S_6(\mathcal{E})$ at 70 since $\mathcal{E}$ does not change. Suppose on the other hand that $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ holds at 68 and suppose that e is the witness. Then we will show that e is also a witness for $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ at 70. In fact, it is easy to see that all conditions of $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ except for $S_9[*.\mathit{find}](e[1])$ follow from $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ at 68 and $P_=(\mathcal{S})$ and $P_\subseteq(\mathcal{F})$ at 113. Now, by $S_9[*.\mathit{find}](e[1])$ at 68, there is a c such that $c \in \delta_\chi(e) \wedge c.\mathit{find}_{68} \not\equiv c$. Also, by $\mathit{hf}(e[1])$ at 68, it follows that $\mathit{hf}(c)$ at 68 (by G_6 at 68). It then follows from G_{19} at 68 that $c \in \mathit{lhs}(\mathcal{S}_{68})$. But since $\mathcal{S} = \mathcal{S}_{68}$ by $P_=(\mathcal{S})$ at 113, it follows by G_{19} at 70 that $c.\mathit{find} \not\equiv c$.

$\underline{S_{11}[\mathcal{S}, *.\mathit{find}](\mathcal{E})}$**:** Suppose $e \in \mathcal{E}$. We must show that $S_1[\mathcal{S}](e[2], e[2]) \wedge \mathit{fr}(e) \wedge S_4[\mathcal{S}, *.\mathit{find}](e)$. $S_1[\mathcal{S}](e[2], e[2])$ follows from $S_{11}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ at 68 and $P_=(\mathcal{S})$ at 113. $\mathit{fr}(e)$ follows from $P_{16}[*.\mathit{find}](X, \mathcal{E})$ at 70 which we showed above. Finally, consider $S_4[\mathcal{S}, *.\mathit{find}](e)$. Let $t \trianglelefteq e$ and suppose $\neg \mathit{hf}(t)$. We must show that $s \equiv \mathit{canon}_\chi(\mathcal{S}(t))$. But by $P_\subseteq(\mathcal{F})$ at 113, $\neg \mathit{hf}(t)$ at 68, so $s \equiv \mathit{canon}_\chi(\mathcal{S}_{68}(t))$, from which it follows (by $P_=(\mathcal{S})$ at 113) that $s \equiv \mathit{canon}_\chi(\mathcal{S}(t))$.

$\underline{\mathcal{A} = \mathcal{A}^* \cup X}$**:** By $\mathcal{A} = \mathcal{A}^* \cup X_{68}$ at 68, $\mathcal{A} = \mathcal{A}' \cup \{e\}$ at 113, and $X = X_{68} \cup \{e\}$.

$\underline{S_{13}[*.find](\mathcal{A}^*)}$**:** By $S_{13}[*.find](\mathcal{A}^*)$ at 68 and $P_{\subseteq}(\mathcal{F})$ at 113.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$**:** We know that, $\mathcal{M}_{68} \subseteq \mathcal{R}_{68}$, so by $P_{\subseteq}(\mathcal{R})$ at 113, $\mathcal{M}_{68} \subseteq \mathcal{R}$. But by $\mathcal{M} \subseteq \mathcal{R}$ at 113, we also know that $(\mathcal{M} - \mathcal{M}_{68}) \subseteq \mathcal{R}$.

Line 72:

```
...   [ A* := A; Φ* := Φ; N* := N; X := ∅ ]
66.   G, P=(all), P4[Φ,H,I,N](E), ¬I, free(E) ⊆ V, P12[Φ,H](E), false ∉ E,
      ∀e ∈ E. fr(e), P13(E), S3[A,I,*.find], S6(E) ∨ S10[S,*.find](E),
      S11[S,*.find](E)
67.   FOREACH e ∈ E DO BEGIN
...
70.     G, P=(H,S), P⊆(V,F,R), P15[Φ,N](Φ*,N*,X), P13(E), X ⊆ E,
        P16[*.find](X,E), P17[Φ](Φ* ∪ X), free(E) ⊆ V', S6(E) ∨ S10[S,*.find](E),
        S11[S,*.find](E), A = A* ∪ X, S13[*.find](A*), M ⊆ R
71.   END
72.   G, P=(S), P⊆(F,R), P1[Φ,H,I,N], P13(E), P18[*.find](E), A = A* ∪ E,
      S6(E) ∨ S10[S,*.find](E), S11[S,*.find](E), S13[*.find](A*), M ⊆ R, E ⊆ Φ
```

$\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$**:** By $\neg\mathcal{I}$ and $P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](\mathcal{E})$ at 66, $\mathcal{T} \cup \mathcal{N}^* \cup \Phi^* \cup \mathcal{E} \models \mathcal{H}_{66}$. From 66, $\mathcal{E} = \emptyset$, $\mathcal{N}^* = \mathcal{N}$, $\Phi^* = \Phi$, and $\mathcal{H}_{66} = \mathcal{H}$, so it follows that $\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H}$. From 70, by $P_{15}[\Phi, \mathcal{N}](\Phi^*, \mathcal{N}^*, X)$ at 70 and the end-of-loop condition $X = \mathcal{E}$, it follows that $\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H}_{66}$. Then, by $P_=(\mathcal{H})$ at 70 and 66, $\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H}$.

$\underline{P_{18}[*.find](\mathcal{E})}$**:** From 66: trivial since $\mathcal{E} = \emptyset$. From 70: by $P_{16}[*.find](X, \mathcal{E})$ and the end-of-loop condition $X = \mathcal{E}$.

$\underline{S_{13}[*.find](\mathcal{A}^*)}$**:** From 66: by $S_3[\mathcal{A}, \mathcal{I}, *.find]$, $\neg\mathcal{I}$, and $\mathcal{A}^* = \mathcal{A}$ at 66. From 70: trivial.

$\underline{\mathcal{E} \subseteq \Phi}$**:** From 66: trivial since $\mathcal{E} = \emptyset$. From 70: by $P_{15}[\Phi, \mathcal{N}](\Phi^*, \mathcal{N}^*, X)$ and the end-of-loop condition $X = \mathcal{E}$.

Line 76:

```
73.   [ X := ∅; R* := R; S* := S ]
74.   G, P_=(S), P_⊆(F,R), P_1[Φ,H,I,N], P_13(E), P_18[*.find](E), A = A* ∪ E,
      S_6(E) ∨ S_10[S,*.find](E), S_11[S,*.find](E), S_13[*.find](A*), M ⊆ R, E ⊆ Φ
75.   FOREACH e ∈ E DO BEGIN
76.     G, P_⊆(F,lhs(S)), P_1[Φ,H,I,N], P_13(E), E ⊆ Φ, P_19[*.find](X,E),
        A = A* ∪ E, S_13[*.find](A* ∪ X), S_14[*.find](R*,X), lhs(X) ⊆ R,
        S_15[S,*.find](E,X), S_16[S,*.find](R*,E,X,S*), S_17(E,X), e ∈ E, e ∉ X
...
78.     G, P_⊆(F,lhs(S)), P_1[Φ,H,I,N], P_13(E), E ⊆ Φ, P_19[*.find](X,E),
        A = A* ∪ E, S_13[*.find](A* ∪ X), S_14[*.find](R*,X), lhs(X) ⊆ R,
        S_15[S,*.find](E,X), S_16[S,*.find](R*,E,X,S*), S_17(E,X)
79.   END
```

Other than the trivial property $e \in \mathcal{E}$, the properties at 76 and 78 are identical and none of them depend on e, so it suffices to consider only the transition from 74.

$\underline{P_{19}[*.find](X,\mathcal{E})}$**:** By $P_{18}[*.find](\mathcal{E})$ at 74 and the fact that $X = \emptyset$.

$\underline{S_{14}[*.find](\mathcal{R}^*,X)}$**:** $\mathcal{M} \subseteq \mathcal{R}$ at 74, so $\mathcal{M} - \mathcal{R}^* = \emptyset$.

$\underline{lhs(X) \subseteq \mathcal{R}}$**:** Trivial since $X = \emptyset$.

$\underline{S_{15}[\mathcal{S},*.find](\mathcal{E},X)}$**:** If $S_{10}[\mathcal{S},*.find](\mathcal{E})$ at 74 then we also have $S_{10}[\mathcal{S},*.find](\mathcal{E})$ at 76, from which $S_{15}[\mathcal{S},*.find](\mathcal{E},X)$ follows easily. Otherwise, $S_6(\mathcal{E})$ by $S_6(\mathcal{E}) \vee S_{10}[\mathcal{S},*.find](\mathcal{E})$ at 74. It then remains to show that $\forall\, e \in \mathcal{E}.\mathcal{S}(e) \equiv e$ and $\forall\, e \in \mathcal{E}.S_1[\mathcal{S}](e[2],e[2])$. Consider $e \in \mathcal{E}$. $S_1[\mathcal{S}](e[2],e[2])$ follows from $S_{11}[\mathcal{S},*.find](\mathcal{E})$ at 74. We now show that $\mathcal{S}(e) \equiv e$. First, we know that $e[1]$ is a χ-leaf by $S_6(\mathcal{E})$. Also, $e[1].find \equiv e[1]$ by $P_{18}[*.find](\mathcal{E})$ at 74. It follows by G_{19} that $e[1] \notin lhs(\mathcal{S})$, so $\mathcal{S}(e[1]) \equiv e[1]$. Now, we know that $e[2] \equiv canon_\chi(\mathcal{S}(e[2]))$ by $S_{11}[\mathcal{S},*.find](\mathcal{E})$ at 74. Thus, by Lemma A.1, $\mathcal{S}(e[2]) \equiv e[2]$.

$\underline{S_{16}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E},X,\mathcal{S}^*)}$**:** Consider $t \in \mathcal{R}^*$. Since at this point $X = \emptyset$, $\mathcal{R} = \mathcal{R}^*$, and $\mathcal{S} = \mathcal{S}^*$, we know that $hf(t)$, $find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \equiv canon_\chi(\mathcal{S}(t))$, and $\delta_\chi(canon_\chi(\mathcal{S}(t))) \cap lhs(X) = \emptyset$. This is sufficient to satisfy the property $S_{16}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E},X,\mathcal{S}^*)$.

$\underline{S_{17}(\mathcal{E}, X)}$: Trivial since $X = \emptyset$.

Line 78:

76. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))\ ,\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]\ ,\ P_{13}(\mathcal{E})\ ,\ \mathcal{E} \subseteq \Phi\ ,\ P_{19}[*.find](X, \mathcal{E})\ ,$
$\mathcal{A} = \mathcal{A}^* \cup \mathcal{E}\ ,\ S_{13}[*.find](\mathcal{A}^* \cup X)\ ,\ S_{14}[*.find](\mathcal{R}^*, X)\ ,\ lhs(X) \subseteq \mathcal{R}\ ,$
$S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)\ ,\ S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)\ ,\ S_{17}(\mathcal{E}, X)\ ,\ e \in \mathcal{E}\ ,\ e \notin X$

77. $e[1].find$:= $e[2]$; [$X := X \cup \{e\}$; IF $\neg S_8(e[1])$ THEN $\mathcal{S} := \{e\}(\mathcal{S}) \cup \{e\}$;]

78. $\underline{G},\ \underline{P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))},\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]\ ,\ P_{13}(\mathcal{E})\ ,\ \mathcal{E} \subseteq \Phi\ ,\ \underline{P_{19}[*.find](X, \mathcal{E})}\ ,$
$\mathcal{A} = \overline{\mathcal{A}^* \cup \mathcal{E}}\ ,\ \underline{S_{13}[*.find](\mathcal{A}^* \cup X)}\ ,\ \underline{S_{14}[*.find](\mathcal{R}^*, X)}\ ,\ \overline{lhs(X) \subseteq \mathcal{R}}\ ,$
$\underline{S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)}\ ,\ \underline{S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)}\ ,\ \underline{S_{17}(\mathcal{E}, X)}$

$\underline{G}$: We consider the global properties that depend on *find* or $\mathcal{S}$. First note that $HF_{76} = HF$. This is because $e[1]$ is the only expression whose find pointer has changed, but $hf(e[1])$ at 76 by $P_{19}[*.find](X, \mathcal{E})$ since $e \in \mathcal{E}$.

$\underline{G_2}$: Since $e[1].find_{76} \equiv e[1]$ and $e[2].find_{76} \equiv e[2]$ (by $P_{19}[*.find](X, \mathcal{E})$, $e \in \mathcal{E}$, and $e \notin X$ at 76), the result of executing line 77 is to merge the $\sim$-equivalence classes whose representatives are $e[1]$ and $e[2]$ respectively. It is not hard to see, then, that $\mathcal{F}$ is the symmetric-transitive closure of $(\mathcal{F}_{76} \cup \{e\})$. Now, $\mathcal{T} \cup \Phi \models \mathcal{F}_{76}$ by G_2 at 76. Also, $e \in \Phi$ by $\mathcal{E} \subseteq \Phi$ and $e \in \mathcal{E}$. Thus, since $\mathcal{T}$ includes the properties of equality, $\mathcal{T} \cup \Phi \models \mathcal{F}$.

$\underline{G_5}$: Since G_5 holds at 76 and since (as we showed above) $e[1].find_{76} \equiv e[1]$ and $e[2].find_{76} \equiv e[2]$, setting $e[1].find$ to $e[2].find$ cannot cause $find^*(d)$ to be undefined for any expression d for which $find^*(d)$ is defined at 76.

$\underline{G_6}$: By G_6 at 76 and $HF_{76} = HF$.

$\underline{G_7}$: By G_7 at 76 and $HF_{76} = HF$.

$\underline{G_8}$: As shown above, $\mathcal{F}$ is the symmetric-transitive closure of $(\mathcal{F}_{76} \cup \{e\})$. But since $e[1].find_{76} \equiv e[1]$ and $e[2].find_{76} \equiv e[2]$, $(e[1] = e[1]) \in \mathcal{F}_{76}$ and $(e[2] = e[2]) \in \mathcal{F}_{76}$. Thus, $free(\mathcal{F}_{76}) = free(\mathcal{F})$. G_8 then follows by G_8 at 76.

$\underline{G_9}$: The above argument also shows that there are no new terms in $\mathcal{F}$ that were not in $\mathcal{F}_{76}$, so G_9 follows from G_9 at 76.

$\underline{G_{11}}$: By G_{11} at 76 and $HF_{76} = HF$.

$\underline{G_{13}}$: Suppose $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ at 76. Then $e[1]$ is a compound χ-term, so $\mathcal{S}$ is unchanged. Thus, $\mathcal{S}$ is in χ-solved form by G_{13} at 76. If $\neg S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ at 76, then by $S_{15}[\mathcal{S}, *.\mathit{find}](\mathcal{E}, X)$ at 76, $\mathcal{E}$ is in χ-solved form and $\mathcal{S}_{76}(e) \equiv e$. It is thus easy to see that since $\mathcal{S}_{76}$ is in χ-solved form (by G_{13} at 76), $\{e\}(\mathcal{S}_{76}) \cup \{e\}$ is also in χ-solved form.

$\underline{G_{14}}$: As described above, $\mathcal{F}$ is the symmetric-transitive closure of $\mathcal{F}_{76} \cup \{e\}$. Thus, in particular, $\mathcal{F} \models \mathcal{F}_{76} \cup \{e\}$. By G_{14} at 76, $\mathcal{T} \cup \mathcal{F}_{76} \models \mathcal{S}_{76}$, and finally, by definition, $\mathcal{T} \cup \mathcal{S}_{76} \cup \{e\} \models \mathcal{S}$. Putting these together, we get $\mathcal{T} \cup \mathcal{F} \models \mathcal{S}$.

$\underline{G_{15}}$: We first show that $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(e)$. If $e[1]$ is a χ-leaf, then this is trivial since $e \in \mathcal{S}$. Otherwise, $e[1]$ is a compound χ-term, and thus $\mathcal{E}$ is not in χ-solved form. Then, by $S_{15}[\mathcal{S}, *.\mathit{find}](\mathcal{E}, X)$ at 76, $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ must be true at 76 which implies that $e[2] \equiv \mathit{canon}_\chi(\mathcal{S}(e[1]))$, from which it follows that $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(e[1] = e[2])$. Next, we note that by definition, $\gamma_\chi(\mathcal{S}) \models \gamma_\chi(\mathcal{S}_{76})$, so it follows that $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(\mathcal{F}_{76})$ by G_{15} at 76. But $\mathcal{F}$ is the symmetric-transitive closure of $\mathcal{F}_{76} \cup \{e\}$. Thus, since $\mathcal{T}_\chi$ includes symmetry and transitivity of equality, $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(\mathcal{F})$.

$\underline{G_{16}}$: By G_{16} at 76 and $\mathit{HF}_{76} = \mathit{HF}$.

$\underline{G_{17}}$: By G_{17} at 76 and $\mathit{HF}_{76} = \mathit{HF}$.

$\underline{G_{18}}$: A little thought reveals that for this property to false, there would have to be a compound χ-term t such that $\mathit{hf}(t) \wedge t.\mathit{find} \not\equiv t$ and $c.\mathit{find} \equiv c$ for all $c \in \delta_\chi(t)$. By G_{18} at 76, there is no such term t at 76. There are two ways that the execution of line 77 could create such a term:

1. The first way is if $e[1]$ is the term t. For this to be the case, $e[1]$ would have to be a compound χ-term with $c.\mathit{find}_{76} \equiv c$ for all $c \in \delta_\chi(e[1])$. But, by $S_{15}[\mathcal{S}, *.\mathit{find}](\mathcal{E}, X)$ at 76, either $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ or $S_6(\mathcal{E})$ must hold at 76. If $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ holds, then the last condition of $S_{10}[\mathcal{S}, *.\mathit{find}](\mathcal{E})$ states that $\exists c \in \delta_\chi(e).\ (c.\mathit{find}_{76} \not\equiv c)$, which contradicts our assumption. If, on the other hand, $S_6(\mathcal{E})$, then $e[1]$ must be a χ-leaf, which also contradicts an assumption.

2. The other possibility is that $e[1]$ is a χ-leaf of some term t where $t.find_{76} \not\equiv t$ and $e[1] \equiv e[2]$. Again, by $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$ at 76, either $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ or $S_6(\mathcal{E})$ must hold at 76. If $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ holds, then $e[1]$ is not a χ-leaf which contradicts our assumption, and if $S_6(\mathcal{E})$ holds at 76, then we cannot have $e[1] \equiv e[2]$ since then $\mathcal{E}$ would not be in χ-solved form.

$\underline{G_{19}}$**:** Suppose $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76. Then $e[1]$ is a compound χ-term, so $\mathcal{S}$ is unchanged. Also, the *find* attributes of no χ-leaves are changed and $HF_{76} = HF$, so G_{19} follows from G_{19} at 76. If $\neg S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76, then by $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$ at 76, $\mathcal{E}$ is in χ-solved form and $\mathcal{S} = \{e\}(\mathcal{S}_{76}) \cup \{e\}$ (which is in χ-solved form as shown above). Thus $lhs(\mathcal{S}) = lhs(\mathcal{S}_{76}) \cup \{e[1]\}$. Now, note that for $t \not\equiv e[1]$, $t.find_{76} \equiv t.find$ and $t \in lhs(\mathcal{S}_{76}) \leftrightarrow t \in lhs(\mathcal{S})$. For $t = e[1]$, clearly $t \in lhs(Eq)$, but since $e[1] \not\equiv e[2]$ (otherwise $\mathcal{E}$ would not be in χ-solved form), $t.find \not\equiv t$. Finally, the only terms in $\mathcal{S}$ that are not in $\mathcal{S}_{76}$ are those in e, but $hf(t)$ for every sub-term t of e by $P_{19}[*.find](X, \mathcal{E})$ at G_6 at 76 and $HF = HF_{76}$. G_{19} thus follows from G_{19} at 76.

$\underline{P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))}$**:** As described above, $\mathcal{F}$ is the symmetric-transitive closure of $\mathcal{F}_{76} \cup \{e\}$, so clearly $\mathcal{F}_{76} \subseteq \mathcal{F}$. $\mathcal{F}' \subseteq \mathcal{F}$ then follows from $P_{\subseteq}(\mathcal{F})$ at 76. To show that $lhs(\mathcal{S}_{76}) \subseteq lhs(\mathcal{S})$, assume that $\neg S_8(e[1])$ (otherwise, $\mathcal{S} = \mathcal{S}_{76}$, so it is trivial) and note that as shown above in the proof of G_{13}, $\mathcal{S}_{76}(e) \equiv e$. Thus, $\delta_\chi(e) \cap lhs(\mathcal{S}_{76}) = \emptyset$, and so $lhs(\{e\}(\mathcal{S}_{76})) = lhs(\mathcal{S}_{76})$. Then, since $\mathcal{S} = \{e\}(\mathcal{S}_{76}) \cup \{e\}$, it follows that $lhs(\mathcal{S}_{76}) \subseteq lhs(\mathcal{S})$.

$\underline{P_{19}[*.find](X, \mathcal{E})}$**:** Follows easily by $P_{19}[*.find](X, \mathcal{E})$ at 76, the execution of line 77, and $X = X_{76} \cup \{e\}$.

$\underline{S_{13}[*.find](\mathcal{A}^* \cup X)}$**:** We must show $\forall\, e \in X.\ (Op(e) = \text{`='} \rightarrow e[1] \sim e[2])$. Suppose $d \in X$. If $d \in X_{76}$, then $Op(e) = \text{`='} \rightarrow e[1] \sim e[2]$ by $S_{13}[*.find](\mathcal{A}^* \cup X)$ at 76 and $P_{\subseteq}(\mathcal{F})$ (shown above). Otherwise, $d \equiv e$, but $e[1] \sim e[2]$ by the execution of line 77 and the definition of $\sim$.

$\underline{S_{14}[*.find](\mathcal{R}^*, X)}$**:** We must show $(\mathcal{M} - \mathcal{R}^*) \subseteq lhs(X)$. Suppose $t \in (\mathcal{M} - \mathcal{R}^*)$. If $t \in (\mathcal{M}_{76} - \mathcal{R}^*)$, then $t \in lhs(X)$ by $S_{14}[*.find](\mathcal{R}^*, X)$ at 76 and $X_{76} \subseteq X$. Otherwise, since $\mathcal{M} = \mathcal{M}_{76} \cup \{e[1]\}$, we must have $t \equiv e[1]$. But $e[1] \in lhs(X)$ since $X = X_{76} \cup \{e\}$.

$\underline{S_{17}(\mathcal{E}, X)}$**:** Clearly, $X \subseteq \mathcal{E}$ since $X_{76} \subseteq \mathcal{E}$ by $S_{17}(\mathcal{E}, X)$ at 76, $e \in \mathcal{E}$, and $X = X_{76} \cup \{e\}$. Now, suppose that $\mathcal{E}$ is in χ-solved form. Then $X_{76} \subseteq \mathcal{S}$ by $S_{17}(\mathcal{E}, X)$ at 76. Also, $e[1]$ is a χ-leaf, so $\mathcal{S} = \{e\}(\mathcal{S}_{76}) \cup \{e\}$. Now, $\{e\}(X_{76}) = X_{76}$ since $X_{76} \subseteq \mathcal{E}$ and $e \in \mathcal{E}$ and $\mathcal{E}$ is in χ-solved form. Thus we have $\mathcal{S} = \{e\}(\mathcal{S}_{76}) \cup \{e\}$, $X_{76} \subseteq \mathcal{S}$, $\{e\}(X_{76}) = X_{76}$, and $X = X_{76} \cup \{e\}$. It follows that $X \subseteq \mathcal{S}$.

$\underline{S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)}$**:** We must show that $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ or $[S_6(\mathcal{E}) \wedge (\forall\, e \in (\mathcal{E} - X).\, \mathcal{S}(e) \equiv e) \wedge (\forall\, e \in \mathcal{E}.\, S_1[\mathcal{S}](e[2], e[2]))]$.

1. Suppose $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76. Then we can show that $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ also holds at 78. Let e be the witness for $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76. We will show that e is the witness at 78 as well. Clearly, $\mathcal{E} = \{e\} \wedge S_8(e[1])$ at 78 follows from $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76. Also, $hf(e[1])$ follows since, as has been shown above, $HF_{76} = HF$. Then, since $e[1]$ is a compound χ-term, $\mathcal{S}$ is unchanged from 76, so $e[2] \equiv canon_\chi(\mathcal{S}(e[1]))$. Finally, the only expression whose *find* attribute changed from 76 to 78 was not a χ-leaf, so $\exists\, c \in \delta_\chi(e).\, (c.find \not\equiv c)$ must hold at 78 since it holds at 76.

2. Suppose on the other hand that $\neg S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76. Then it follows from $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$ at 76 that $S_6(\mathcal{E})$ holds, $\forall\, e \in (\mathcal{E} - X_{76}).\mathcal{S}_{76}(e) \equiv e$, and $\forall\, e \in \mathcal{E}.\, e[2] \equiv canon_\chi(\mathcal{S}_{76}(e[2]))$. We must show that these conditions also hold at 78.

   $S_6(\mathcal{E})$**:** By $S_6(\mathcal{E})$ at 76.

   $\forall\, e \in (\mathcal{E} - X).\, \mathcal{S}(e) \equiv e$**:** Consider $d \in (\mathcal{E} - X)$. Since $X = X_{76} \cup \{e\}$, we know that $d \in (\mathcal{E} - X_{76})$, so $\mathcal{S}_{76}(d) \equiv d$ and thus $\delta_\chi(d) \cap lhs(\mathcal{S}_{76}) = \emptyset$. Also, because $\mathcal{E}$ is in χ-solved form and $e \in \mathcal{E}$ (and $e \not\equiv d$ since $e \in X$), $e[1] \notin \delta_\chi(d)$. Now, as we showed above, (in the proof of G_{19}),

$lhs(\mathcal{S}) = lhs(\mathcal{S}_{76}) \cup \{e[1]\}$, so it follows that $\delta_\chi(d) \cap lhs(\mathcal{S}) = \emptyset$, and thus $\mathcal{S}(d) \equiv d$.

$\forall\, e \in \mathcal{E}.\, e[2] \equiv canon_\chi(\mathcal{S}(e[2]))$**:** Let $d \in \mathcal{E}$. From $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$ at 76, we know that $d[2] \equiv canon_\chi(\mathcal{S}_{76}(d[2]))$, so $\mathcal{S}_{76}(d) \equiv d$ by Lemma A.1. But as shown above, $lhs(\mathcal{S}) = lhs(\mathcal{S}_{76}) \cup \{e[1]\}$ and since $\mathcal{E}$ is in χ-solved form, $e[1] \notin \delta_\chi(d[2])$, so $\mathcal{S}(d[2]) \equiv d[2]$. Thus, $d[2] \equiv canon_\chi(\mathcal{S}(d[2]))$.

$\underline{lhs(X) \subseteq \mathcal{R}}$**:** Consider $d \in X$. In order to show $d \in \mathcal{R}$, we must show $hf(d[1]) \wedge find^*(d[1]) \equiv canon_\chi(\mathcal{S}(d[1]))$. We know that $d \in \mathcal{E}$ by $S_{17}(\mathcal{E}, X)$ at 78 (shown above). Then, by $P_{19}[*.find](X, \mathcal{E})$ at 78 (also shown above), $d[1].find \equiv d[2]$, so $hf(d[1]) \wedge find^*(d[1]) \equiv d[2]$. Now, if $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 78, then we know that $d[2] \equiv canon_\chi(\mathcal{S}(d[1]))$. Otherwise, $\mathcal{E}$ is in χ-solved form (by $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$ at 78, shown above), so again by $S_{17}(\mathcal{E}, X)$, $d \in \mathcal{S}$. It then follows by G_{13} that $\mathcal{S}(d[1]) \equiv d[2]$. But, by $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$, $d[2] \equiv canon_\chi(\mathcal{S}(d[2]))$, so by property 2 of *canon*, $canon_\chi(d[2]) \equiv d[2]$, and thus $canon_\chi(\mathcal{S}(d[1])) \equiv d[2]$.

$\underline{S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)}$**:** Suppose $t \in \mathcal{R}^*$. We know that $hf(t)$ since $hf(t)$ at 76 (by $S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)$ at 76) and $HF = HF_{76}$. Now, we must consider three cases.

1. Suppose $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) = \emptyset \vee \neg S_6(\mathcal{E})$. We must show that $find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \equiv canon_\chi(\mathcal{S}(t))$. Since $lhs(X_{76}) \subseteq lhs(X)$, it must be the case that $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X_{76}) = \emptyset \vee \neg S_6(\mathcal{E})$. Thus, by $S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)$, $find^*_{76}(t) \equiv canon_\chi(\mathcal{S}^*(t)) \equiv canon_\chi(\mathcal{S}_{76}(t))$. We consider two sub-cases.

   (a) Suppose $S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76. Then since e must be a compound χ-term, we know that $\mathcal{S}_{76} = \mathcal{S}$, and thus it follows trivially that $canon_\chi(\mathcal{S}_{76}(t)) \equiv canon_\chi(\mathcal{S}(t))$. Now, if $find^*(t) \equiv find^*_{76}(t)$, then clearly $find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \equiv canon_\chi(\mathcal{S}(t))$. If, on the other hand, $find^*(t) \not\equiv find^*_{76}(t)$, then it must be the case that $find^*_{76}(t) \equiv e[1]$ and $find^*(t) \equiv e[2]$. But then $e[1] \equiv canon_\chi(\mathcal{S}_{76}(t)) \equiv canon_\chi(\mathcal{S}(t))$.

Furthermore, $e[2] \equiv canon_\chi(\mathcal{S}(e[1]))$ by $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$ (shown above), so $e[2] \equiv canon_\chi(\mathcal{S}(t))$ by Corollary A.1, and thus $e[1] \equiv e[2]$ which contradicts the assumption that $find^*(t) \not\equiv find^*_{76}(t)$.

(b) Suppose $\neg S_{10}[\mathcal{S}, *.find](\mathcal{E})$ at 76. Then $\mathcal{E}$ is in χ-solved form (by $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$) and thus $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) = \emptyset$. But $e \in X$ and $e[1]$ is a χ-leaf, so in particular $canon_\chi(\mathcal{S}^*(t)) \not\equiv e[1]$. Thus, since $find^*_{76}(t) \equiv canon_\chi(\mathcal{S}^*(t))$, it follows that $find^*(t) \equiv find^*_{76}(t)$. Thus, $find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \equiv canon_\chi(\mathcal{S}_{76}(t))$. Finally, we show $canon_\chi(\mathcal{S}_{76}(t)) \equiv canon_\chi(\mathcal{S}(t))$. First note that $\mathcal{S}(canon_\chi(\mathcal{S}_{76}(t))) \equiv canon_\chi(\mathcal{S}_{76}(t))$. To show this, suppose it is not true. Then there would have to be some $c \in \delta_\chi(canon_\chi(\mathcal{S}_{76}(t)))$ such that $c \in lhs(\mathcal{S})$. But $\mathcal{S} = \{e\}(\mathcal{S}_{76}) \cup \{e\}$, so $c \in \mathcal{S}_{76}$ or $c \equiv e[1]$. But if $c \in lhs(\mathcal{S}_{76})$, then $c \notin \delta_\chi(canon_\chi(\mathcal{S}_{76}(t)))$ by property 3 of *canon*, which is a contradiction. Also, $c \equiv e[1]$ would imply $c \notin \delta_\chi(canon_\chi(\mathcal{S}_{76}(t)))$ because $canon_\chi(\mathcal{S}_{76}(t)) \equiv canon_\chi(\mathcal{S}^*(t))$, $e \in X$ and $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) = \emptyset$. Thus $\mathcal{S}(canon_\chi(\mathcal{S}_{76}(t))) \equiv canon_\chi(\mathcal{S}_{76}(t))$, and so it then follows from property 2 of *canon* that $canon_\chi(\mathcal{S}(canon_\chi(\mathcal{S}_{76}(t)))) \equiv canon_\chi(\mathcal{S}_{76}(t))$. But by Corollary A.1, $canon_\chi(\mathcal{S}(canon_\chi(\mathcal{S}_{76}(t)))) \equiv canon_\chi(\mathcal{S}(t))$.

2. Suppose that $\mathcal{E}$ is in χ-solved form, $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) \neq \emptyset$, and $canon_\chi(\mathcal{S}^*(t))$ is a χ-leaf. We must show that $find^*(t) \equiv canon_\chi(\mathcal{S}(t)) \wedge \delta_\chi(canon_\chi(\mathcal{S}(t))) \cap lhs(\mathcal{E}) = \emptyset$. Since $canon_\chi(\mathcal{S}^*(t))$ is a χ-leaf, it follows by $S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)$ at 76 that $find^*_{76}(t) \equiv canon_\chi(\mathcal{S}_{76}(t))$. There are two cases.

   (a) Suppose $canon_\chi(\mathcal{S}^*(t)) \equiv e[1]$. Note that $X = X_{76} \cup \{e\}$ and the left-hand sides of X are all distinct (since $X \subseteq \mathcal{E}$ and $\mathcal{E}$ is in χ-solved form), so $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X_{76}) = \emptyset$. It then follows from $S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)$ at 76 that $canon_\chi(\mathcal{S}^*(t)) \equiv canon_\chi(\mathcal{S}_{76}(t))$. Now, since $find^*_{76}(t) \equiv canon_\chi(\mathcal{S}_{76}(t))$, it follows that $find^*_{76}(t) \equiv canon_\chi(\mathcal{S}^*(t))$ and thus $find^*_{76}(t) \equiv e[1]$. So, after the execution of line 77, $find^*(t) \equiv e[2]$. But we know that $\mathcal{S}(canon_\chi(\mathcal{S}^*(t))) \equiv e[2]$

since $canon_\chi(\mathcal{S}^*(t)) \equiv e[1]$, $e \in X$ and $X \subseteq \mathcal{S}$ (by $S_{17}(\mathcal{E}, X)$ at 78, shown above). Finally, we also know that $e[2] \equiv canon_\chi(\mathcal{S}(e[2]))$ by $S_{15}[\mathcal{S}, *.find](\mathcal{E}, X)$ (also shown above), so by substitution, we have that $e[2] \equiv canon_\chi(\mathcal{S}(\mathcal{S}(canon_\chi(\mathcal{S}^*(t)))))$ which further reduces to $canon_\chi(\mathcal{S}(canon_\chi(\mathcal{S}^*(t))))$ which, finally, is equivalent to $canon_\chi(\mathcal{S}(t))$ by Corollary A.1. Thus, $find^*(t) \equiv canon_\chi(\mathcal{S}(t))$. It is then easy to see that $\delta_\chi(canon_\chi(\mathcal{S}(t))) \cap lhs(\mathcal{E}) = \emptyset$ since $canon_\chi(\mathcal{S}(t)) \equiv e[2]$, $e \in \mathcal{E}$, and $\mathcal{E}$ is in χ-solved form.

(b) Suppose that $canon_\chi(\mathcal{S}^*(t)) \not\equiv e[1]$. Then, since $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) \neq \emptyset$, it must be the case that $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X_{76}) \neq \emptyset$ (since, as mentioned above, $X = X_{76} \cup \{e\}$ and the left-hand sides of X are all distinct). Thus, by $S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)$ at 76, $\delta_\chi(canon_\chi(\mathcal{S}_{76}(t))) \cap lhs(\mathcal{E}) = \emptyset$. In particular, we have that $e[1] \notin \delta_\chi(canon_\chi(\mathcal{S}_{76}(t)))$, and so, since $find^*_{76}(t) \equiv canon_\chi(\mathcal{S}_{76}(t))$, it follows that $find^*(t) \equiv canon_\chi(\mathcal{S}_{76}(t))$. Also, since $\mathcal{S} = \{e\}(\mathcal{S}_{76}) \cup \{e\}$, it follows by property 3 of *canon* that $\mathcal{S}(canon_\chi(\mathcal{S}_{76}(t))) \equiv canon_\chi(\mathcal{S}_{76}(t))$. It then follows that $canon_\chi(\mathcal{S}(canon_\chi(\mathcal{S}_{76}(t)))) \equiv canon_\chi(\mathcal{S}_{76}(t))$ by property 2 of *canon*. But we also know from Corollary A.1 that $canon_\chi(\mathcal{S}(canon_\chi(\mathcal{S}_{76}(t)))) \equiv canon_\chi(\mathcal{S}(t))$, so we can conclude that $find^*(t) \equiv canon_\chi(\mathcal{S}_{76}(t)) \equiv canon_\chi(\mathcal{S}(t))$. Finally, it is easy to see that $\delta_\chi(canon_\chi(\mathcal{S}(t))) \cap lhs(\mathcal{E}) = \emptyset$, since $\delta_\chi(canon_\chi(\mathcal{S}_{76}(t))) \cap lhs(\mathcal{E}) = \emptyset$ and $canon_\chi(\mathcal{S}_{76}(t)) \equiv canon_\chi(\mathcal{S}(t))$.

3. Finally, suppose $\mathcal{E}$ is in χ-solved form, $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) \neq \emptyset$, and $canon_\chi(\mathcal{S}^*(t))$ is a compound χ-term. We must show that $find^*(t) \equiv canon_\chi(\mathcal{S}^*(t))$. Since $canon_\chi(\mathcal{S}^*(t))$ is a compound χ-term, it follows from $S_{16}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E}, X, \mathcal{S}^*)$ at 76 that $find^*_{76}(t) \equiv canon_\chi(\mathcal{S}^*(t))$. But, it also follows that $canon_\chi(\mathcal{S}^*(t)) \not\equiv e[1]$ since $e[1]$ must be a χ-leaf (because $e \in \mathcal{E}$ and $\mathcal{E}$ is in χ-solved form), so $find^*(t) \equiv find^*_{76}(t) \equiv canon_\chi(\mathcal{S}^*(t))$.

Line 80:

```
73.    [ X := ∅; R* := R; S* := S ]
74.    G, P=(S), P⊆(F,R), P1[Φ,H,I,N], P13(E), P18[*.find](E), A = A* ∪ E,
       S6(E) ∨ S10[S,*.find](E), S11[S,*.find](E), S13[*.find](A*), M ⊆ R, E ⊆ Φ
75.    FOREACH e ∈ E DO BEGIN
...
78.      G, P⊆(F,lhs(S)), P1[Φ,H,I,N], P13(E), E ⊆ Φ, P19[*.find](X,E),
         A = A* ∪ E, S13[*.find](A* ∪ X), S14[*.find](R*,X), lhs(X) ⊆ R,
         S15[S,*.find](E,X), S16[S,*.find](R*,E,X,S*), S17(E,X)
79.    END
80.    G, P⊆(F,lhs(S)), P1[Φ,H,I,N], S3[A,I,*.find], lhs(E) ⊆ R,
       S16[S,*.find](R*,E,E,S*), S18[S,*.find](R*), S20[S](E)
```

$S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$**:** By $S_{13}[*.\mathit{find}](\mathcal{A}^* \cup X)$ (at both 74 and 78) and the end-of-loop condition, $X = \mathcal{E}$, we have $S_{13}[*.\mathit{find}](\mathcal{A}^* \cup \mathcal{E})$. But $\mathcal{A} = \mathcal{A}^* \cup \mathcal{E}$, so $S_{13}[*.\mathit{find}](\mathcal{A})$, from which $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$ follows.

$S_{16}[\mathcal{S},*.\mathit{find}](\mathcal{R}^*,\mathcal{E},\mathcal{E},\mathcal{S}^*)$**:** From 74: Consider $t \in \mathcal{R}^*$. Since at this point $X = \emptyset$, $\mathcal{R} = \mathcal{R}^*$, and $\mathcal{S} = \mathcal{S}^*$, we know that $\mathit{hf}(t)$, $\mathit{find}^*(t) \equiv \mathit{canon}_\chi(\mathcal{S}^*(t)) \equiv \mathit{canon}_\chi(\mathcal{S}(t))$, and $\delta_\chi(\mathit{canon}_\chi(\mathcal{S}(t))) \cap \mathit{lhs}(X) = \emptyset$. Since we also know that $\mathcal{E} = \emptyset$, this is sufficient to satisfy the property $S_{16}[\mathcal{S},*.\mathit{find}](\mathcal{R}^*,\mathcal{E},\mathcal{E},\mathcal{S}^*)$. From 78: follows from $S_{16}[\mathcal{S},*.\mathit{find}](\mathcal{R}^*,\mathcal{E},X,\mathcal{S}^*)$ at 78 by the end-of-loop condition $X = \mathcal{E}$.

$S_{18}[\mathcal{S},*.\mathit{find}](\mathcal{R}^*)$**:** From 74: by $\mathcal{M} \subseteq \mathcal{R}$. From 78: by $S_{14}[*.\mathit{find}](\mathcal{R}^*,X)$ at 78 and the end-of-loop condition, $(\mathcal{M} - \mathcal{R}^*) \subseteq \mathit{lhs}(\mathcal{E})$. But $\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R}$, so $(\mathcal{M} - \mathcal{R}^*) \subseteq \mathcal{R}$.

$S_{20}[\mathcal{S}](\mathcal{E})$**:** From 74: trivial since $\mathcal{E} = \emptyset$. From 78: suppose $S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E})$ at 78. Then there exists e such that $\mathcal{E} = \{e\}$ and e is a compound χ-term. Suppose on the other hand that $\neg S_{10}[\mathcal{S},*.\mathit{find}](\mathcal{E})$ at 78. Then by $S_{15}[\mathcal{S},*.\mathit{find}](\mathcal{E},X)$ at 78, $\mathcal{E}$ is in χ-solved form, so by $S_{17}(\mathcal{E},X)$ at 78 and the end-of-loop condition, $\mathcal{E} \subseteq \mathcal{S}$.

Line 82:

80. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))\ ,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]\ ,\ S_3[\mathcal{A},\mathcal{I},*.find]\ ,\ lhs(\mathcal{E}) \subseteq \mathcal{R},$
$S_{16}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E},\mathcal{E},\mathcal{S}^*)\ ,\ S_{18}[\mathcal{S},*.find](\mathcal{R}^*)\ ,\ S_{20}[\mathcal{S}](\mathcal{E})$
81. $[\ X\ :=\ \emptyset;\ ]$
82. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))\ ,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]\ ,\ S_3[\mathcal{A},\mathcal{I},*.find]\ ,\ lhs(\mathcal{E}) \subseteq \mathcal{R},$
$S_{18}[\mathcal{S},*.find](\mathcal{R}^*)\ ,\ \underline{S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E}-X,\mathcal{S}^*)}\ ,\ S_{20}[\mathcal{S}](\mathcal{E})$

$\underline{S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E}-X,\mathcal{S}^*)}$**:** Note that $X = \emptyset$, so it suffices to show the property $S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E},\mathcal{S}^*)$. Consider $t \in \mathcal{R}^*$. By $S_{16}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E},\mathcal{E},\mathcal{S}^*)$ at 80, we know that $hf(t)$. Also, $find^*(t) \equiv canon_\chi(\mathcal{S}(t))$ and thus $t \in \mathcal{R}$ unless $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(\mathcal{E}) \neq \emptyset \wedge S_8(canon_\chi(\mathcal{S}^*(t)))$, in which case $find^*(t) \equiv canon_\chi(\mathcal{S}^*(t))$. $S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E},\mathcal{S}^*)$ follows easily.

Line 86:

84. $G, P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})), P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}], S_3[\mathcal{A},\mathcal{I},*.find], \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ e \in \mathcal{E},$
$\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ \mathcal{I} \vee S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E}-X,\mathcal{S}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ e \notin X$
85. $\mathcal{L}$:= $e[1].notify$; $[\ \mathcal{U}\ :=\ \emptyset;\ ]$
86. $G, P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})), P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}], S_3[\mathcal{A},\mathcal{I},*.find], \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ e \in \mathcal{E},$
$\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ \underline{S_{21}[*.find](\mathcal{L},e[1])},\ e \notin X,$
$\underline{S_{22}[\mathcal{I},\mathcal{S},*.find](e,\mathcal{R}^*,\mathcal{E}-X,\mathcal{L}-\mathcal{U},\mathcal{S}^*)}$

$\underline{S_{21}[*.find](\mathcal{L},e[1])}$**:** By G_{17}.

$\underline{S_{22}[\mathcal{I},\mathcal{S},*.find](e,\mathcal{R}^*,\mathcal{E}-X,\mathcal{L}-\mathcal{U},\mathcal{S}^*)}$**:** Suppose $\neg\mathcal{I}$ and let $t \in \mathcal{R}^*$. Since we know that $\mathcal{I} \vee S_{19}[\mathcal{S},*.find](\mathcal{R}^*,\mathcal{E}-X,\mathcal{S}^*)$ at 84, $t \in \mathcal{R}$ or $S_8(canon_\chi(\mathcal{S}^*(t))) \wedge hf(t) \wedge find^*(t) \equiv canon_\chi(\mathcal{S}^*(t)) \wedge \delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(X) \neq \emptyset$. It remains to show that if $t \notin \mathcal{R}$, then $e[1] \in \delta_\chi(canon_\chi(\mathcal{S}^*(t))) \rightarrow (\chi, canon_\chi(\mathcal{S}^*(t))) \in (\mathcal{L}-\mathcal{U})$. But this follows by G_{16}, $\mathcal{L} = e[1].notify$, and $\mathcal{U} = \emptyset$.

Line 90:

88. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),$ $\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ S_{21}[*.find](\mathcal{L},e[1]),\ e \in \mathcal{E},\ e \notin X,$ $S_{22}[\mathcal{I},\mathcal{S},*.find](e,\mathcal{R}^*,\mathcal{E}-X,\mathcal{L}-\mathcal{U},\mathcal{S}^*),\ (i,d) \in \mathcal{L},\ (i,d) \notin \mathcal{U}$

89. $\texttt{TheoryUpdate}_i\texttt{(}e\texttt{,}d\texttt{);}\ [\ \mathcal{U}\ \texttt{:=}\ \mathcal{U}\ \cup\ \{(i,d)\}\texttt{;}\ ]$

90. $\underline{G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),}$ $\underline{\mathcal{I} \vee S_{18}[\mathcal{S},*.find](\mathcal{R}^*),\ S_{20}[\mathcal{S}](\mathcal{E}),\ S_{21}[*.find](\mathcal{L},e[1]),\ e \in \mathcal{E},\ e \notin X,}$ $\underline{S_{22}[\mathcal{I},\mathcal{S},*.find](e,\mathcal{R}^*,\mathcal{E}-X,\mathcal{L}-\mathcal{U},\mathcal{S}^*)}$

...

270. $G,\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \underline{S_{28}[*.find](i,d)}$

271. $\texttt{TheoryUpdate}_i\texttt{(}e\texttt{,}d\texttt{)}$

272. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),\ \mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R}),$ $\mathcal{I}' \to \mathcal{I}$

...

379. $\texttt{END TheoryUpdate}_\chi$

380. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ S_3[\mathcal{A},\mathcal{I},*.find],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),\ \mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R}),$ $\mathcal{I}' \to \mathcal{I},\ S_{34}[\mathcal{I},\mathcal{S},*.find](d)$

First, consider the preconditions of $\texttt{TheoryUpdate}_i$.

$\underline{S_{28}[*.find](i,d)}$**:** Suppose $i = \chi$. Since $(i,d) \in \mathcal{L}$ and by $S_{21}[*.find](\mathcal{L},e[1])$ at 88, we have $hf(d)$ and $S_8(d)$ and $e[1] \in \delta_\chi(d)$. Thus, it remains to show that $\exists c \in \delta_\chi(d).\ (c.find \not\equiv c)$. But $e[1] \in \delta_\chi(d)$ and since $e[1]$ must therefore be a χ-leaf, it follows from $S_{20}[\mathcal{S}](\mathcal{E})$ (and $e \in \mathcal{E}$) at 88 that $e[1] \in lhs(\mathcal{S})$. Thus, by G_{19}, $e[1].find \not\equiv e[1]$.

Now consider the properties at 90. Note that $\neg\mathcal{I}$ implies $\neg\mathcal{I}_{88}$ since $\mathcal{I}' \to \mathcal{I}$ at 272.

$\underline{G}$**:** By G at 272.

$\underline{P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))}$**:** By $P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S}))$ at 88 and 272.

$\underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]}$**:** By $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ at 272.

$\underline{S_3[\mathcal{A},\mathcal{I},*.find]}$**:** By $S_3[\mathcal{A},\mathcal{I},*.find]$ at 272.

$\underline{\mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R})}$**:** Suppose $\neg\mathcal{I}$. Then $\neg\mathcal{I}_{88}$, so $lhs(\mathcal{E}) \subseteq \mathcal{R}_{88}$. Then, since $\mathcal{R}' \subseteq \mathcal{R}$ at 272, $lhs(\mathcal{E}) \subseteq \mathcal{R}$.

$\underline{\mathcal{I} \vee S_{18}[\mathcal{S}, *.find](\mathcal{R}^*)}$**:** Suppose $\neg\mathcal{I}$. Then $\neg\mathcal{I}_{88}$, so $\mathcal{M}_{88} - \mathcal{R}^* \subseteq \mathcal{R}_{88}$. But $\mathcal{R}_{88} \subseteq \mathcal{R}$ (by $\mathcal{R}' \subseteq \mathcal{R}$ at 272), so $\mathcal{M}_{88} - \mathcal{R}^* \subseteq \mathcal{R}$. Finally, $\mathcal{M}_{272} \subseteq \mathcal{R}$, so $\mathcal{M} - \mathcal{M}_{88} \subseteq \mathcal{R}$ at 90, and thus $\mathcal{M} - \mathcal{R}^* \subseteq \mathcal{R}$.

$\underline{S_{20}[\mathcal{S}](\mathcal{E})}$**:** By $S_{20}[\mathcal{S}](\mathcal{E})$ at 88 and $P_{\subseteq}(lhs(\mathcal{S}))$ at 272.

$\underline{S_{21}[*.find](\mathcal{L}, e[1])}$**:** By $S_{21}[*.find](\mathcal{L}, e[1])$ at 88 and $P_{\subseteq}(\mathcal{F})$ at 272.

$\underline{e \in \mathcal{E}}$**:** By $e \in \mathcal{E}$ at 88.

$\underline{e \notin X}$**:** By $e \notin X$ at 88.

$\underline{S_{22}[\mathcal{I}, \mathcal{S}, *.find](e, \mathcal{R}^*, \mathcal{E} - X, \mathcal{L} - \mathcal{U}, \mathcal{S}^*)}$**:** Suppose $\neg\mathcal{I}$, so that $\neg\mathcal{I}_{88}$. Then suppose $t \in \mathcal{R}^*$. If $t \in \mathcal{R}_{88}$, then by $\mathcal{R}' \subseteq \mathcal{R}$ at 272, $t \in \mathcal{R}$. Otherwise, for convenience let $s \equiv canon_\chi(\mathcal{S}^*(t))$ and note that by $S_{22}[\mathcal{I}, \mathcal{S}, *.find](e, \mathcal{R}^*, \mathcal{E} - X, \mathcal{L} - \mathcal{U}, \mathcal{S}^*)$ at 88, we have $hf_{88}(t)$, $find^*_{88}(t) \equiv s$ (and thus $s.find_{88} \equiv s$), s is a compound χ-term, $\delta_\chi(s) \cap lhs(\mathcal{E} - X_{88}) \neq \emptyset$, and $e[1] \in \delta_\chi(s) \rightarrow (\chi, s) \in (\mathcal{L} - \mathcal{U}_{88})$. We consider two cases.

1. Suppose $(i, d) = (\chi, s)$. We first show that $s \in \mathcal{R}$. First note that since $i = \chi$, we know that $S_{34}[\mathcal{I}, \mathcal{S}, *.find](d)$ at 272, so (since $d \equiv s$) $(s.find \not\equiv s) \vee (s \in \mathcal{R})$. Now suppose that $s \notin \mathcal{R}$ and $(s.find \not\equiv s)$. But recall that $s.find_{88} \equiv s$, so this means that $s \in \mathcal{M}_{272}$, and thus $s \in \mathcal{R}$ since $\mathcal{M}_{272} \subseteq \mathcal{R}$. Now, if $s \in \mathcal{R}$, then $find^*(s) \equiv canon_\chi(\mathcal{S}(s))$. But $s \equiv canon_\chi(\mathcal{S}^*(t))$, so $find^*(s) \equiv canon_\chi(\mathcal{S}(canon_\chi(\mathcal{S}^*(t))))$, and thus $find^*(s) \equiv canon_\chi(\mathcal{S}(t))$ by Corollary A.1. But since $find^*_{88}(t) \equiv s$, it follows by $P_{\subseteq}(\mathcal{F})$ at 272 that $find^*(s) \equiv find^*(t)$, and so $find^*(t) \equiv canon_\chi(\mathcal{S}(t))$, and thus $t \in \mathcal{R}$.
2. Suppose $(i, d) \neq (\chi, s)$. If $s.find \not\equiv s$, then using the same reasoning as in the previous case, we can conclude that $s \in \mathcal{R}$ and thus $t \in \mathcal{R}$. Suppose on the other hand that $s.find \equiv s$. We already know that s is a compound χ-term. Now, since $find^*_{88}(t) \equiv s$, it follows by $P_{\subseteq}(\mathcal{F})$ at 272 that $hf(t)$ and $find^*(t) \equiv find^*(s) \equiv s$. Also, X is unchanged from 88, so $\delta_\chi(s) \cap lhs(\mathcal{E} - X) \neq \emptyset$. It remains to show that $e[1] \in \delta_\chi(s) \rightarrow (\chi, s) \in$

$(\mathcal{L} - \mathcal{U})$. But this follows trivially since $e[1] \in \delta_\chi(s) \rightarrow (\chi, s) \in (\mathcal{L} - \mathcal{U}_{88})$, $\mathcal{U} = \mathcal{U}_{88} \cup \{(i,d)\}$, and $(\chi, s) \neq (i,d)$.

Line 92:

86. G, $P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\mathcal{I} \vee (\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R})$, $e \in \mathcal{E}$, $\mathcal{I} \vee S_{18}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*)$, $S_{20}[\mathcal{S}](\mathcal{E})$, $S_{21}[*.\mathit{find}](\mathcal{L}, e[1])$, $e \notin X$, $S_{22}[\mathcal{I}, \mathcal{S}, *.\mathit{find}](e, \mathcal{R}^*, \mathcal{E} - X, \mathcal{L} - \mathcal{U}, \mathcal{S}^*)$
87. `FOREACH` $(i,d) \in \mathcal{L}$ `DO BEGIN`
...
90. G, $P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\mathcal{I} \vee (\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R})$, $\mathcal{I} \vee S_{18}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*)$, $S_{20}[\mathcal{S}](\mathcal{E})$, $S_{21}[*.\mathit{find}](\mathcal{L}, e[1])$, $e \in \mathcal{E}$, $e \notin X$, $S_{22}[\mathcal{I}, \mathcal{S}, *.\mathit{find}](e, \mathcal{R}^*, \mathcal{E} - X, \mathcal{L} - \mathcal{U}, \mathcal{S}^*)$
91. `END` [X := $X \cup \{e\}$;]
92. G, $P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\mathcal{I} \vee (\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R})$, $\mathcal{I} \vee S_{18}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*)$, $\underline{\mathcal{I} \vee S_{19}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*, \mathcal{E} - X, \mathcal{S}^*)}$, $S_{20}[\mathcal{S}](\mathcal{E})$

$\underline{\mathcal{I} \vee S_{19}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*, \mathcal{E} - X, \mathcal{S}^*)}$**:** Suppose $\neg\mathcal{I}$ and let $t \in \mathcal{R}^*$. Again, for convenience, let $s \equiv \mathit{canon}_\chi(\mathcal{S}^*(t))$. If $t \notin \mathcal{R}$, then by $S_{22}[\mathcal{I}, \mathcal{S}, *.\mathit{find}](e, \mathcal{R}^*, \mathcal{E} - X, \mathcal{L} - \mathcal{U}, \mathcal{S}^*)$ at 86 or 90, we have s is a compound χ-term, $\mathit{hf}(t)$, $\mathit{find}^*(t) \equiv s$, $\delta_\chi(s) \cap \mathit{lhs}(\mathcal{E} - X_{90}) \neq \emptyset$, and $e[1] \in \delta_\chi(s) \rightarrow (\chi, s) \in (\mathcal{L} - \mathcal{U})$. It only remains to show $\delta_\chi(s) \cap \mathit{lhs}(\mathcal{E} - X) \neq \emptyset$. Suppose the contrary. The only way this could happen is if $e[1] \in \delta_\chi(s)$. But this implies that $(\chi, s) \in (\mathcal{L} - \mathcal{U})$, and by the end-of-loop condition, $\mathcal{L} = \mathcal{U}$, so this is impossible.

Line 94:

81. [X := $\emptyset$;]
82. G, $P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R}$, $S_{18}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*)$, $S_{19}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*, \mathcal{E} - X, \mathcal{S}^*)$, $S_{20}[\mathcal{S}](\mathcal{E})$
83. `FOREACH` $e \in \mathcal{E}$ `DO BEGIN`
...
92. G, $P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\mathcal{I} \vee (\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R})$, $\mathcal{I} \vee S_{18}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*)$, $\mathcal{I} \vee S_{19}[\mathcal{S}, *.\mathit{find}](\mathcal{R}^*, \mathcal{E} - X, \mathcal{S}^*)$, $S_{20}[\mathcal{S}](\mathcal{E})$
93. `END`
94. G, $P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S}))$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$, $\underline{\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})}$, $\mathcal{I} \vee (\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R})$, $\underline{\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})}$

$\underline{\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})}$**:** Assume $\neg\mathcal{I}$, and suppose $t \in \mathcal{R}^*$. By $S_{19}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E} - X, \mathcal{S}^*)$ at 82 or $\mathcal{I} \vee S_{19}[\mathcal{S}, *.find](\mathcal{R}^*, \mathcal{E} - X, \mathcal{S}^*)$ at 92, either $t \in \mathcal{R}$ or $\delta_\chi(canon_\chi(\mathcal{S}^*(t))) \cap lhs(\mathcal{E} - X) \neq \emptyset$. But by the end-of-loop condition, $\mathcal{E} = X$, so we must have $t \in \mathcal{R}$.

$\underline{\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})}$**:** Assume $\neg\mathcal{I}$. Then by $S_{18}[\mathcal{S}, *.find](\mathcal{R}^*)$ at 82 or $\mathcal{I} \vee S_{18}[\mathcal{S}, *.find](\mathcal{R}^*)$ at 92, $\mathcal{M} - \mathcal{R}^* \subseteq \mathcal{R}$. But as just shown, $\mathcal{R}^* \subseteq \mathcal{R}$, so $\mathcal{M} \subseteq \mathcal{R}$.

A.5.5 AssertFormula

Line 101:

99. G, $free(e) \subseteq \mathcal{V}$, $fr(e)$, $P_{11}(e)$, $P_{20}[\mathcal{H}](\Phi, e)$, $S_4[\mathcal{S}, *.find](e)$
100. `AssertFormula(`e`)` [Z := $\emptyset$; G_{10}^{ok} := FALSE;]
101. G, $P_=(all)$, $free(e) \subseteq \mathcal{V}'$, $fr(e)$, $P_{11}(e)$, $\underline{P_{17}[\Phi](\Phi')}$, $P_{20}[\mathcal{H}](\Phi', e)$, $\underline{P_{21}[*.find]}$, $\underline{P_{38}[\mathcal{A}, \Lambda_*](e)}$, $S_4[\mathcal{S}, *.find](e)$

$\underline{P_{17}[\Phi](\Phi')}$**:** Trivial since $\Phi = \Phi'$.

$\underline{P_{21}[*.find]}$**:** Trivial since $e.find \equiv e.find'$ for all e.

$\underline{P_{38}[\mathcal{A}, \Lambda_*](e)}$**:** By G_{10} at 99, if $t \in \Lambda_j$ and $j \neq \mathcal{T}(t)$, then t occurs j-alien in some sub-term of $\mathcal{A}$. Also, it follows by G_{11} at 99 that $hf(t)$.

Line 103:

... [Z := $\emptyset$;]
101. G, $P_=(all)$, $free(e) \subseteq \mathcal{V}'$, $fr(e)$, $P_{11}(e)$, $P_{17}[\Phi](\Phi')$, $P_{20}[\mathcal{H}](\Phi', e)$, $P_{21}[*.find]$, $P_{38}[\mathcal{A}, \Lambda_*](e)$, $S_4[\mathcal{S}, *.find](e)$
102. `FOREACH maximal sub-term` t `of` e `DO BEGIN`
103. G, $P_=(\mathcal{A}, \mathcal{H}, \mathcal{S})$, $P_\subseteq(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R})$, $free(e) \subseteq \mathcal{V}'$, $fr(e)$, $P_{11}(e)$, $P_{17}[\Phi](\Phi')$, $P_{20}[\mathcal{H}](\Phi', e)$, $P_{21}[*.find]$, $\underline{P_{22}[*.find](Z)}$, $\underline{P_{23}(t)}$, $\underline{P_{24}[\Lambda_*](Z, e)}$, $P_{38}[\mathcal{A}, \Lambda_*](e)$, $S_4[\mathcal{S}, *.find](e)$, $\underline{\mathcal{M} \subseteq \mathcal{R}}$, $t \lhd e$
...
105. G, $P_=(\mathcal{A}, \mathcal{H}, \mathcal{S})$, $P_\subseteq(\Phi, \mathcal{V}, \mathcal{N}, \mathcal{F}, \mathcal{R})$, $free(e) \subseteq \mathcal{V}'$, $fr(e)$, $P_{11}(e)$, $P_{17}[\Phi](\Phi')$, $P_{20}[\mathcal{H}](\Phi', e)$, $P_{21}[*.find]$, $P_{22}[*.find](Z)$, $P_{24}[\Lambda_*](Z, e)$, $P_{38}[\mathcal{A}, \Lambda_*](e)$, $S_4[\mathcal{S}, *.find](e)$, $\mathcal{M} \subseteq \mathcal{R}$
106. `END`

$\underline{P_{22}[*.find](Z)}$**:** From 101: $Z = \emptyset$. From 105: follows trivially from $P_{22}[*.find](Z)$ at 105.

$\underline{P_{23}(t)}$**:** By the loop condition.

$\underline{P_{24}[\Lambda_*](Z,e)}$**:** From 101: $Z = \emptyset$. From 105: follows trivially from $P_{24}[\Lambda_*](Z,)$ at 105.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$**:** From 101: by $P_{=}(all)$ at 101. From 105: follows trivially from $\mathcal{M} \subseteq \mathcal{R}$ at 105.

Line 105:

```
102.   FOREACH maximal sub-term t of e DO BEGIN
```
103. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R}),\ free(e) \subseteq \mathcal{V}',\ fr(e),\ P_{11}(e),\ P_{17}[\Phi](\Phi'),\ P_{20}[\mathcal{H}](\Phi',e),\ P_{21}[*.find],\ P_{22}[*.find](Z),\ P_{23}(t),\ P_{24}[\Lambda_*](Z,e),\ P_{38}[\mathcal{A},\Lambda_*](e),\ S_4[\mathcal{S},*.find](e),\ \mathcal{M} \subseteq \mathcal{R},\ t \lhd e$

```
104.   SetupTerm(t,T(e)); [ Z := Z ∪ {t}; ]
```
105. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R}),\ free(e) \subseteq \mathcal{V}',\ fr(e),\ P_{11}(e),\ P_{17}[\Phi](\Phi'),\ P_{20}[\mathcal{H}](\Phi',e),\ P_{21}[*.find],\ P_{22}[*.find](Z),\ P_{24}[\Lambda_*](Z,e),\ P_{38}[\mathcal{A},\Lambda_*](e),\ S_4[\mathcal{S},*.find](e),\ \mathcal{M} \subseteq \mathcal{R}$

```
...
```
114. $G,\ \underline{fr(t)},\ P_{23}(t),\ \underline{free(t) \subseteq \mathcal{V}},\ \underline{S_4[\mathcal{S},*.find](t)}$

```
115. SetupTerm(t,i)
...
145. END SetupTerm
```
146. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),\ P_{26}[\Lambda_*](t,i),\ P_{27}[*.find](t),\ P_{28}[*.find](t),\ t.find \equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ \mathcal{M} \subseteq \mathcal{R}$

First we consider the preconditions of `SetupTerm`.

$\underline{fr(t)}$**:** By $fr(e)$ and G_7 at 103, the definition of fr, and the fact that t is a *maximal* sub-term of e.

$\underline{free(t) \subseteq \mathcal{V}}$**:** By $free(e) \subseteq \mathcal{V}'$, $P_{\subseteq}(\mathcal{V})$, and $t \lhd e$ at 103.

$\underline{S_4[\mathcal{S},*.find](t)}$**:** By $S_4[\mathcal{S},*.find](e)$ at 103, $t \lhd e$.

Now we consider the properties at line 105.

$\underline{G}$**:** By G at 146.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})}$**:** By $P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})$ at 146.

$\underline{P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R})}$**:** By $P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R})$ at 146.

$\underline{\mathit{free}(e) \subseteq \mathcal{V}'}$**:** By $\mathit{free}(e) \subseteq \mathcal{V}'$ at 103.

$\underline{\mathit{fr}(e)}$**:** By $\mathit{fr}(e)$, $t \lhd e$ at 103, $P_{27}[*.\mathit{find}](t)$ and $P_{28}[*.\mathit{find}](t)$ at 146, and the definition of fr.

$\underline{P_{11}(e)}$**:** By $P_{11}(e)$ at 103.

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{17}[\Phi](\Phi')$ at 103, we have $\mathcal{T} \cup \Phi' \models \exists\, \overline{x}.\, \Phi_{103}$ and $\overline{x} \cap \mathcal{V}' = \emptyset$, where $\overline{w} = \mathit{free}(\Phi_{103}) - \mathit{free}(\Phi')$ Then, by $P_{17}[\Phi](\Phi')$ at 146, we have $\mathcal{T} \cup \Phi_{103} \models \exists\, \overline{y}.\, \Phi$ and $\overline{y} \cap \mathcal{V}_{103} = \emptyset$, where $\overline{y} = \mathit{free}(\Phi) - \mathit{free}(\Phi_{103})$. By $P_{\subseteq}(\Phi)$ at 103 and 146 and Lemma A.4 it follows that $\mathcal{T} \cup \Phi' \models \exists\, \overline{w}.\Phi$, where $\overline{w} = \overline{x} \cup \overline{y} = \mathit{free}(\Phi) - \mathit{free}(\Phi')$. It then follows easily from $P_{\subseteq}(\mathcal{V})$ at 146 that $\overline{w} \cap \mathcal{V}' = \emptyset$.

$\underline{P_{20}[\mathcal{H}](\Phi', e)}$**:** By $P_{20}[\mathcal{H}](\Phi', e)$ at 103.

$\underline{P_{21}[*.\mathit{find}]}$**:** By $P_{21}[*.\mathit{find}]$ at 103 and $P_{27}[*.\mathit{find}](t)$ and $P_{28}[*.\mathit{find}](t)$ at 146.

$\underline{P_{22}[*.\mathit{find}](Z)}$**:** By $P_{22}[*.\mathit{find}](Z)$ at 103, $\forall\, t \in Z_{103}.\, t.\mathit{find}_{103} \equiv t$. It then follows by $P_{27}[*.\mathit{find}](t)$ and $P_{28}[*.\mathit{find}](t)$ at 146 that $\forall\, t \in Z_{103}.\, t.\mathit{find} \equiv t$. Finally, since $Z = Z_{103} \cup \{t\}$, and $t.\mathit{find} \equiv t$ at 146, we have $\forall\, t \in Z.\, t.\mathit{find} \equiv t$.

$\underline{P_{24}[\Lambda_*](Z, e)}$**:** Suppose $s \trianglelefteq Z$, $s \lhd d \trianglelefteq e$, and s occurs $\mathcal{T}(d)$-alien in d. We must show $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$. We consider four cases.

1. Suppose $s \trianglelefteq Z_{103}$. $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$ follows easily by $P_{24}[\Lambda_*](Z, e)$ at 103.
2. Suppose $s \not\trianglelefteq Z_{103}$ and $d \trianglelefteq t$. Then we have $s \trianglelefteq t$ and $s \lhd d \trianglelefteq t$, so $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$ follows by $P_{24}[\Lambda_*](t, t)$ at 146.
3. Suppose $s \not\trianglelefteq Z_{103}$, $d \not\trianglelefteq t$, and $\mathtt{s} \lhd t$. Then by the definition of alien, s occurs $\mathcal{T}(t)$-alien in t, and thus $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$ follows again by $P_{24}[\Lambda_*](t, t)$ at 146.

4. Suppose $s \not\trianglelefteq Z_{103}$, $d \not\trianglelefteq t$, and $s \not\trianglelefteq t$. We know $s \trianglelefteq Z$ and $Z = Z_{103} \cup \{t\}$, so we must have $s \trianglelefteq t$, and thus $s \equiv t$. Now, $s \triangleleft d \trianglelefteq d$ and e is a literal (by $P_{11}(e)$), so by the definition of literal, we must have $e \equiv d$ or $e \equiv \neg d$. In either case, we have $\mathcal{T}(d) = \mathcal{T}(e)$ and thus $\mathcal{T}(t) \neq \mathcal{T}(e)$. Thus, by $P_{26}[\Lambda_*](t,i)$ at 146, $t \in \Lambda_{\mathcal{T}(t)} \wedge t \in \Lambda_{\mathcal{T}(e)}$, and thus $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$.

$\underline{P_{38}[\mathcal{A}, \Lambda_*](e)}$**:** Suppose $s \in \Lambda_j$ and $j \neq \mathcal{T}(s)$. If $s \not\trianglelefteq t$, then by $P_{39}[\Lambda_*](t)$ at 146, $s \in \Lambda_j$ at 103. But then, by $P_{38}[\mathcal{A}, \Lambda_*](e)$ at 103, s occurs j-alien in some sub-term of $\mathcal{A}_{103} \cup \{e\}$. It follows from $P_{=}(()\mathcal{A})$ at 146 that s occurs j-alien in some sub-term of $\mathcal{A} \cup \{e\}$. Suppose on the other hand that $s \trianglelefteq t$. Then by $P_{42}[\Lambda_*](t,i)$ at 146, either $s \in \Lambda_j$ at 103 (the case we just handled above), or $s \equiv t \wedge j = i$ (in which case s occurs j-alien in e), or s occurs j-alien in some sub-term of t, and thus in some sub-term of e.

$\underline{S_4[\mathcal{S}, *.\mathit{find}](e)}$**:** By $S_4[\mathcal{S}, *.\mathit{find}](e)$ at 103, and $P_{=}(\mathcal{S})$ and $P_{\subseteq}(\mathcal{F})$ at 146.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$**:** By $\mathcal{M} \subseteq \mathcal{R}$ at 103 and 146 and $P_{\subseteq}(\mathcal{R})$ at 146.

Line 107:

```
101.   G, P_=(all), free(e) ⊆ V', fr(e), P_11(e), P_17[Φ](Φ'), P_20[H](Φ', e),
       P_21[*.find], P_38[A, Λ_*](e), S_4[S, *.find](e)
102.   FOREACH maximal sub-term t of e DO BEGIN
...
105.     G, P_=(A, H, S), P_⊆(Φ, V, N, F, R), free(e) ⊆ V', fr(e), P_11(e),
         P_17[Φ](Φ'), P_20[H](Φ', e), P_21[*.find], P_22[*.find](Z), P_24[Λ_*](Z, e),
         P_38[A, Λ_*](e), S_4[S, *.find](e), M ⊆ R
106.   END
107.   G, P_=(A, H, S), P_⊆(Φ, V, N, F, R), free(e) ⊆ V', P_11(e), P_17[Φ](Φ'),
       P_20[H](Φ', e), P_21[*.find], P_24[Λ_*](e, e), P_25[*.find](e), P_38[A, Λ_*](e),
       M ⊆ R
```

$\underline{P_{24}[\Lambda_*](e,e)}$**:** From 101: trivial since there are no terms in e. From 105: since the property is true for every sub-term of every maximal sub-term of e by $P_{24}[\Lambda_*](Z,e)$ at 105 (and the end-of-loop condition), it is therefore true for every term in e.

$\underline{P_{25}[*.find](e)}$**:** From 101: trivial since there are no terms in e. From 105: follows by $P_{22}[*.find](Z)$ at 105 and the end-of-loop condition.

Line 109:

107. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R}),\ free(e)\subseteq\mathcal{V}',\ P_{11}(e),\ P_{17}[\Phi](\Phi'),\ P_{20}[\mathcal{H}](\Phi',e),\ P_{21}[*.find],\ P_{24}[\Lambda_*](e,e),\ P_{25}[*.find](e),\ P_{38}[\mathcal{A},\Lambda_*](e),\ \mathcal{M}\subseteq\mathcal{R}$

108. $[\ \mathcal{A}_{\mathcal{T}(e)}$ `:=` $\mathcal{A}_{\mathcal{T}(e)}\ \cup\ \{e\};\ G_{10}^{ok}$ `:= TRUE; ]`

109. $\underline{G},\ P_{=}(\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R}),\ free(e)\subseteq\mathcal{V}',\ P_{11}(e),\ \underline{P_{17}[\Phi](\Phi'\cup\{e\})},\ P_{20}[\mathcal{H}](\Phi',e),\ P_{21}[*.find],\ P_{25}[*.find](e),\ e\in\Phi,\ \mathcal{A}=\mathcal{A}'\cup\{e\},\ \mathcal{M}\subseteq\mathcal{R}$

$\underline{G}$**:** We must consider those global properties that depend on $\mathcal{A}$ or G_{10}^{ok}.

$\underline{G_2}$**:** Follows trivially from G_2 at 107 since $\Phi_{107}\subseteq\Phi$.

$\underline{G_3}$**:** Let $\{W,X,Z\}$ be the partition of Φ' and Y the set containing e which exist by $P_{20}[\mathcal{H}](\Phi',e)$ at 107. Also, by $P_{20}[\mathcal{H}](\Phi',e)$ we have $\mathcal{T}\cup\mathcal{H}\models\exists\overline{w}.\,W$, where $\overline{w}=free(W)-free(\mathcal{H})$, and $\mathcal{T}\cup\mathcal{H}\cup W\models\exists\overline{x},\overline{y}.\,(X\cup Y)$, where $\overline{x}=free(X)-free(\mathcal{H}\cup W)$ and $\overline{y}=free(Y)-free(\mathcal{H}\cup W\cup X)$, so in particular, $\mathcal{T}\cup\mathcal{H}\cup W\models\exists\overline{x},\overline{y_e}.\,(X\cup\{e\})$, where $\overline{y_e}=free(e)-free(\mathcal{H}\cup W\cup X)$. We also have $\mathcal{T}\cup W\cup X\models\exists\overline{z}.\,Z$, where $\overline{z}=free(Z)-free(W\cup X)$ and $\overline{z}\cap free(\mathcal{H}\cup\{e\})=\emptyset$ (and thus $\overline{z}=free(Z)-free(\mathcal{H}\cup W\cup X\cup\{e\})$) Putting these together we get $\mathcal{T}\cup\mathcal{H}\models\exists\overline{w^*}.\,(W\cup X\cup\{e\}\cup Z)$, where $\overline{w^*}=free(W\cup X\cup\{e\}\cup Z)-free(\mathcal{H})$, but $W\cup X\cup Z=\Phi'$, so we have $\mathcal{T}\cup\mathcal{H}\models\exists\overline{w^*}.\,(\Phi'\cup\{e\})$, where $\overline{w^*}=free(\Phi'\cup\{e\})-free(\mathcal{H})$. Now, by $P_{17}[\Phi](\Phi')$ at 107 (and the fact that $\Phi=\Phi_{107}\cup\{e\}$), $\mathcal{T}\cup\Phi'\models\exists\overline{x^*}.\,(\Phi-\{e\})$ where $\overline{x^*}=free(\Phi-\{e\})-free(\Phi')$ and $\overline{x^*}\cap\mathcal{V}'=\emptyset$. Now, since $free(\mathcal{H})\subseteq\mathcal{V}'$ (by G_8 at 99 and $P_{=}(\mathcal{H})$ at 111) and $free(e)\subseteq\mathcal{V}'$ at 111, we also have $\overline{x^*}=free(\Phi-\{e\})-free(\mathcal{H}\cup\Phi'\cup\{e\})$. We can thus conclude $\mathcal{T}\cup\mathcal{H}\models\exists\overline{y^*}.\,(\Phi'\cup\{e\}\cup(\Phi-\{e\}))$, where $\overline{y^*}=free((\Phi-\{e\})\cup\Phi'\cup\{e\})-free(\mathcal{H})$. Since $\Phi'\subseteq\Phi$ by $P_{\subseteq}(\Phi)$, this simplifies to $\mathcal{T}\cup\mathcal{H}\models\exists\overline{y^*}.\,\Phi$, where $\overline{y^*}=free(\Phi)-free(\mathcal{H})$.

$\underline{G_4}$**:** Follows trivially from G_2 at 107 since $\Phi_{107}\subseteq\Phi$.

$\underline{G_8}$**:** Follows from G_8, $free(e)\subseteq\mathcal{V}'$, and $P_{\subseteq}(\mathcal{V})$ at 107.

$\underline{G_9}$: Suppose t occurs j-alien in some sub-term of $\Phi \cup \mathcal{F}$. We must show $t \in \Lambda_j \wedge t \in \Lambda_{\mathcal{T}(t)}$. First, if $t \in \Phi_{107} \cup \mathcal{F}$, then $t \in \Lambda_j \wedge t \in \Lambda_{\mathcal{T}(t)}$ by G_9 at 107. The only other case is if $t \lhd e$. But then $t \in \Lambda_j \wedge t \in \Lambda_{\mathcal{T}(t)}$ by $P_{24}[\Lambda_*](e, e)$ at 107.

$\underline{G_{10}}$: By $P_{38}[\mathcal{A}, \Lambda_*](e)$ at 107 and $\mathcal{A} = \mathcal{A}_{107} \cup \{e\}$.

$\underline{G_{11}}$: Follows from G_{11}, $P_{25}[*.\mathit{find}](e)$, and G_6 at 107, and the fact that $\mathcal{A} = \mathcal{A}_{107} \cup \{e\}$.

$\underline{G_{20}}$: By G_{20} and $P_{11}(e)$ at 107, and the fact that $\mathcal{A} = \mathcal{A}_{107} \cup \{e\}$.

$\underline{P_{17}[\Phi](\Phi' \cup \{e\})}$: We must show $\mathcal{T} \cup \Phi' \cup \{e\} \models \exists \overline{w}.\, \Phi$ and $\overline{w} \cap \mathcal{V}' = \emptyset$, where $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\Phi' \cup \{e\})$. Now, by $P_{17}[\Phi](\Phi')$ at 107, we have $\mathcal{T} \cup \Phi' \models \exists \overline{w}.\, (\Phi - \{e\})$ and $\overline{w} \cap \mathcal{V}' = \emptyset$, where $\overline{w} = \mathit{free}(\Phi - \{e\}) - \mathit{free}(\Phi')$. It follows that $\mathcal{T} \cup \Phi' \cup \{e\} \models [(\exists \overline{w}.\, (\Phi - \{e\})) \wedge e]$. But $\mathit{free}(e) \subseteq \mathcal{V}'$, $\mathit{free}(e) \cap \overline{w} = \emptyset$, so we can rewrite this as $\mathcal{T} \cup \Phi' \cup \{e\} \models \exists \overline{w}.\, \Phi$, and we can also rewrite $\overline{w}$ as $\overline{w} = \mathit{free}(\Phi) - \mathit{free}(\Phi' \cup \{e\})$.

Line 111:

```
109.  G, P_=(H,S), P_⊆(Φ,V,N,F,R), free(e) ⊆ V', P_17[Φ](Φ' ∪ {e}),
      P_20[H](Φ',e), P_21[*.find], P_25[*.find](e), e ∈ Φ, A = A' ∪ {e}, M ⊆ R
110.  TheoryAssert_T(e)(e);
111.  G, P_=(H,S), P_⊆(Φ,V,N,F,R), P_17[Φ](Φ' ∪ {e}), P_21[*.find],
      P_25[*.find](e), e ∈ Φ, A = A' ∪ {e}, M ⊆ R
...
255. G,
256. TheoryAssert_i(e)
257. G, P_34[all](i)
```

The preconditions of `TheoryAssert` are obvious, so we consider the properties at line 111. For reference, $P_{34}[\mathit{all}](i)$ at 257 implies $P_=(\Phi - \mathcal{B}_i, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, \Lambda_*, *.\mathit{find})$, $P_\subseteq(\mathcal{Q}, \mathcal{B}_i, \mathcal{V}, *.\mathit{notify})$, $\mathcal{T}_i \cup \gamma_i(\Phi_i' \models \exists \overline{w}.\, \mathcal{B}_i) \wedge \overline{w} \cap \mathcal{V}' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$, where $\overline{w} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i')$, and $\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}')) \wedge \mathit{free}(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}$.

$\underline{G}$ By G at 257.

$\underline{P_{=}(\mathcal{H},\mathcal{S})}$**:** By $P_{=}(\mathcal{H},\mathcal{S})$ at 109 and 257.

$\underline{P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R})}$**:** By $P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\mathcal{F},\mathcal{R})$ at 109 and $P_{=}(\mathcal{N},*.\mathit{find})$ and $P_{\subseteq}(\Phi,\mathcal{V})$ at 257.

$\underline{P_{17}[\Phi](\Phi' \cup \{e\})}$**:** By $P_{17}[\Phi](\Phi' \cup \{e\})$ at 109, we have $\mathcal{T} \cup \Phi' \cup \{e\} \models \exists \overline{w}.\, \Phi_{109}$ and $\overline{w} \cap \mathcal{V}\,' = \emptyset$, where $\overline{w} = \mathit{free}(\Phi_{109}) - \mathit{free}(\Phi' \cup \{e\})$. Let Φ_i^* denote the value of Φ_i at 109. Then, by $P_{34}[\mathit{all}](i)$, where $i = \mathcal{T}(e)$, we have $\gamma_i(\mathcal{T}_i \cup \Phi_i^* \models \exists \overline{x}.\, \mathcal{B}_i)$ and $\overline{x} \cap \mathcal{V}_{109} = \emptyset$ (and thus $\overline{x} \cap \mathcal{V}\,' = \emptyset$ by $P_{\subseteq}(\mathcal{V})$ at 109), where $\overline{x} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i^*)$. It follows that $\mathcal{T} \cup \Phi_i^* \models \exists \overline{x}.\, \mathcal{B}_i$, and thus $\mathcal{T} \cup \Phi_{109} \models \exists \overline{x}.\, \mathcal{B}_i$, so $\mathcal{T} \cup \Phi' \cup \{e\} \models \exists \overline{w}, \overline{x}.\, (\Phi_{109} \cup \mathcal{B}_i)$. But $\Phi = \Phi_{109} \cup \mathcal{B}_i$ (by $P_{=}(\Phi - \mathcal{B}_i)$ and $P_{\subseteq}(\mathcal{B}_i)$ at 257), so $\mathcal{T} \cup \Phi' \cup \{e\} \models \exists \overline{w}, \overline{x}.\, \Phi$. As shown, $(\overline{w} \cup \overline{x}) \cap \mathcal{V}\,' = \emptyset$. It remains to show that $\overline{w} \cup \overline{x} = \mathit{free}(\Phi) - \mathit{free}(\Phi' \cup \{e\})$. We know that $\mathit{free}(\Phi) = \mathit{free}(\Phi_{109}) \cup \mathit{free}(\mathcal{B}_i)$ and $\overline{w} = \mathit{free}(\Phi_{109}) - \mathit{free}(\Phi' \cup \{e\})$. Now, $\overline{x} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i^*)$, but we also know that $\overline{x} \cap \mathcal{V}_{109} = \emptyset$, $\mathit{free}(\Phi_{109}) \subseteq \mathcal{V}_{109}$ (by G_8 at 109), and $\mathit{free}(e) \subseteq \mathcal{V}_{109}$ (by $\mathit{free}(e) \subseteq \mathcal{V}\,'$ and $P_{\subseteq}(\mathcal{V})$ at 109), so $\overline{x} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_{109} \cup \{e\})$, and thus $\overline{w} \cup \overline{x} = \mathit{free}(\Phi) - \mathit{free}(\Phi' \cup \{e\})$.

$\underline{P_{21}[*.\mathit{find}]}$**:** By $P_{21}[*.\mathit{find}]$ at 109 and $P_{=}(*.\mathit{find})$ at 257.

$\underline{P_{25}[*.\mathit{find}](e)}$**:** By $P_{25}[*.\mathit{find}](e)$ at 109 and $P_{=}(*.\mathit{find})$ at 257.

$\underline{e \in \Phi}$**:** By $e \in \Phi$ at 109 and $P_{\subseteq}(\Phi)$ at 257.

$\underline{\mathcal{A} = \mathcal{A}' \cup \{e\}}$**:** By $\mathcal{A} = \mathcal{A}' \cup \{e\}$ at 109 and $P_{=}(\Phi - \mathcal{B}_i)$ at 257.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$**:** By $\mathcal{M} \subseteq \mathcal{R}$ at 109 and $P_{=}(*.\mathit{find})$ at 257.

A.5.6 SetupTerm

Line 116:

114. G, $\mathit{fr}(t)$, $P_{23}(t)$, $\mathit{free}(t) \subseteq \mathcal{V}$, $S_4[\mathcal{S}, *.\mathit{find}](t)$
115. `SetupTerm(`t`,`i`)`
116. G, $P_{=}(\mathit{all})$, $\mathit{fr}(t)$, $P_{23}(t)$, $\mathit{free}(t) \subseteq \mathcal{V}$, $\underline{P_{17}[\Phi](\Phi')}$, $S_4[\mathcal{S}, *.\mathit{find}](t)$

$\underline{P_{17}[\Phi](\Phi')}$**:** Trivial since $\Phi' = \Phi$.

Line 120:

118. $G,\ P_{=}(all),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},\ P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,$
$S_4[\mathcal{S}, *.find](t)$
119. $[\ \Lambda_{\mathcal{T}(t)}$:= $\Lambda_{\mathcal{T}(t)} \cup \{t\};\ ]$
120. $\underline{G},\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},$
$P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ \underline{P_{39}[\Lambda_*](t)},\ \underline{P_{42}[\Lambda_*](t,i)},\ S_4[\mathcal{S}, *.find](t)$

$\underline{G}$**:** We must consider those global properties that depend on Λ.

$\underline{G_8}$**:** By G_8 and $free(t) \subseteq \mathcal{V}$ at 118.

$\underline{G_9}$**:** By G_9 at 118 and the fact that nothing was removed from Λ_j for any j.

$\underline{G_{10}}$**:** By definition, $\neg G_{10}^{ok}$ inside of `SetupTerm`.

$\underline{P_{39}[\Lambda_*](t)}$**:** By $P_{=}(all)$ at 118, $P_{39}[\Lambda_*](t)$ holds at 118. Then, since the only change to Λ_j for any j is the addition of t to $\Lambda_{\mathcal{T}(t)}$ and clearly $t \trianglelefteq t$, it follows that $P_{39}[\Lambda_*](t)$ holds at 120.

$\underline{P_{42}[\Lambda_*](t,i)}$**:** By $P_{=}(all)$ at 118, we have $\Lambda_j = \Lambda'_j$ at 118, so $P_{42}[\Lambda_*](t,i)$ holds at 118. Now, suppose $s \trianglelefteq t$, $s \in \Lambda_j$, and $j \neq \mathcal{T}(t)$. If $s \in \Lambda_j$ at 118, then the property holds by $P_{42}[\Lambda_*](t,i)$ at 118. The only other possibility is that $s \equiv t$ and $j = \mathcal{T}(t)$. But we assumed that $j \neq \mathcal{T}(t)$, so this is not possible. Thus $P_{42}[\Lambda_*](t,i)$ holds at 120.

Line 122:

120. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},$
$P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S}, *.find](t)$
121. `TheoryAddSharedTerm`$_{\mathcal{T}(t)}$`(t);`
122. $\underline{G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*),\ fr(t),\ P_{23}(t),\ free(t) \subseteq \mathcal{V},}$
$\underline{P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S}, *.find](t)}$

...

252. $G,\ free(e) \subseteq \mathcal{V}$
253. `TheoryAddSharedTerm`$_i$`(e)`
254. $G,\ P_{34}[all](i)$

The preconditions for `TheoryAddSharedTerm` are trivial, so we consider the properties at line 122. Recall that $P_{34}[all](i)$ at 254 implies $P_{=}(\Phi - \mathcal{B}_i, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, \Lambda_*, *.find)$,

$P_{\subseteq}(\mathcal{Q}, \mathcal{B}_i, \mathcal{V}, *.notify)$, $\mathcal{T}_i \cup \gamma_i(\Phi_i' \models \exists \overline{w}.\, \mathcal{B}_i) \wedge \overline{w} \cap \mathcal{V}\,' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$, where $\overline{w} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i')$, and $\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}')) \wedge \mathit{free}(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}$.

$\underline{G}$**:** By G at 254.

$\underline{P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.\mathit{find})}$**:** By $P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.\mathit{find})$ at 120 and at 254.

$\underline{P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*)}$**:** By $P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*)$ at 120 and $P_{=}(\mathcal{N}, \Lambda_*)$ and $P_{\subseteq}(\Phi, \mathcal{V})$ at 254.

$\underline{\mathit{fr}(t)}$**:** By $\mathit{fr}(t)$ at 120 and $P_{=}(*.\mathit{find})$ at 254.

$\underline{P_{23}(t)}$**:** By $P_{23}(t)$ at 120.

$\underline{\mathit{free}(t) \subseteq \mathcal{V}}$**:** By $\mathit{free}(t) \subseteq \mathcal{V}$ at 120 and $P_{\subseteq}(\mathcal{V})$ at 254.

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{17}[\Phi](\Phi')$ at 120, $\mathcal{T} \cup \Phi' \models \exists \overline{w}.\, \Phi_{120}$ and $\overline{w} \cap \mathcal{V}\,' = \emptyset$, where $\overline{w} = \mathit{free}(\Phi_{120}) - \mathit{free}(\Phi')$. Then, by $P_{34}[\mathit{all}](i)$ at 254, $\mathcal{T} \cup \Phi_{120} \models \exists \overline{x}.\Phi$ and $\overline{x} \cap \mathcal{V}_{120} = \emptyset$, where $\overline{x} = \mathit{free}(\Phi) - \mathit{free}(\Phi_{120})$. Thus, by $P_{\subseteq}(\Phi)$ at 120 and $\Phi = \Phi_{120} \cup \mathcal{B}_i$ at 254, we can use Lemma A.4 to get $\mathcal{T} \cup \Phi' \models \exists \overline{y}.\, \Phi$, where $\overline{y} = \overline{w} \cup \overline{x}\mathit{free}(\Phi) - \mathit{free}(\Phi')$. It follows easily (by $P_{\subseteq}(\mathcal{V})$ at 120) that $\overline{y} \cap \mathcal{V}\,' = \emptyset$.

$\underline{\mathcal{T}(t) \neq i}$**:** By $\mathcal{T}(t) \neq i$ at 120.

$\underline{t \in \Lambda_{\mathcal{T}(t)}}$**:** By $t \in \Lambda_{\mathcal{T}(t)}$ at 120 and $P_{=}(\Lambda_*)$ at 254.

$\underline{P_{39}[\Lambda_*](t)}$**:** By $P_{39}[\Lambda_*](t)$ at 120 and $P_{=}(\Lambda_*)$ at 254.

$\underline{P_{42}[\Lambda_*](t, i)}$**:** By $P_{42}[\Lambda_*](t, i)$ at 120 and $P_{=}(\Lambda_*)$ at 254.

$\underline{S_4[\mathcal{S}, *.\mathit{find}](t)}$**:** By $S_4[\mathcal{S}, *.\mathit{find}](t)$ at 120 and $P_{=}(\mathcal{S}, *.\mathit{find})$ at 254.

Line 124:

122. G, $P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.\mathit{find})$, $P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*)$, $\mathit{fr}(t)$, $P_{23}(t)$, $\mathit{free}(t) \subseteq \mathcal{V}$, $P_{17}[\Phi](\Phi')$, $\mathcal{T}(t) \neq i$, $t \in \Lambda_{\mathcal{T}(t)}$, $P_{39}[\Lambda_*](t)$, $P_{42}[\Lambda_*](t, i)$, $S_4[\mathcal{S}, *.\mathit{find}](t)$
123. [Λ_i := $\Lambda_i \cup \{t\}$;]
124. $\underline{G}$, $P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.\mathit{find})$, $P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*)$, $\mathit{fr}(t)$, $P_{23}(t)$, $\mathit{free}(t) \subseteq \mathcal{V}$, $t \in \Lambda_i$, $P_{17}[\Phi](\Phi')$, $\mathcal{T}(t) \neq i$, $t \in \Lambda_{\mathcal{T}(t)}$, $\underline{P_{39}[\Lambda_*](t)}$, $\underline{P_{42}[\Lambda_*](t, i)}$, $S_4[\mathcal{S}, *.\mathit{find}](t)$

$\underline{G}$: We must consider those global properties that depend on Λ.

$\underline{G_8}$: By G_8 and $\mathit{free}(t) \subseteq \mathcal{V}$ at 122.

$\underline{G_9}$: By G_9 at 122 and the fact that nothing was removed from Λ_j for any j.

$\underline{G_{10}}$: By definition, $\neg G_{10}^{ok}$ inside of `SetupTerm`.

$\underline{P_{39}[\Lambda_*](t)}$: We have $P_{39}[\Lambda_*](t)$ at 122. Then since the only change to Λ_j for any j is the addition of t to $\Lambda_{\mathcal{T}(t)}$ and clearly $t \trianglelefteq t$, it follows that $P_{39}[\Lambda_*](t)$ holds at 124.

$\underline{P_{42}[\Lambda_*](t,i)}$: We have $P_{42}[\Lambda_*](t,i)$ at 122, and the only change from 122 is the addition of t to Λ_i. But this is covered by the case $s \equiv t \wedge j = i$ in $P_{42}[\Lambda_*](t,i)$, so $P_{42}[\Lambda_*](t,i)$ holds at 124.

Line 126:

124. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*),\ \mathit{fr}(t),\ P_{23}(t),\ \mathit{free}(t) \subseteq \mathcal{V},\ t \in \Lambda_i,$
$P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.\mathit{find}](t)$

125. `TheoryAddSharedTerm`$_i$`(`t`);`

126. $\underline{G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*),\ \mathit{fr}(t),\ P_{23}(t),\ \mathit{free}(t) \subseteq \mathcal{V},\ t \in \Lambda_i,}$
$\underline{P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.\mathit{find}](t)}$

This case is almost identical to that for line 122. The only difference is that there is one additional property, $t \in \Lambda_i$ which is preserved trivially by $P_{=}(\Lambda_*)$ at 254.

Line 128:

116. $G,\ P_{=}(\mathit{all}),\ \mathit{fr}(t),\ P_{23}(t),\ \mathit{free}(t) \subseteq \mathcal{V},\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S},*.\mathit{find}](t)$

117. `IF` $\mathcal{T}(t) \neq i$ `THEN BEGIN`

...

126. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*),\ \mathit{fr}(t),\ P_{23}(t),\ \mathit{free}(t) \subseteq \mathcal{V},\ t \in \Lambda_i,$
$P_{17}[\Phi](\Phi'),\ \mathcal{T}(t) \neq i,\ t \in \Lambda_{\mathcal{T}(t)},\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.\mathit{find}](t)$

127. `END`

128. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*),\ \mathit{fr}(t),\ P_{23}(t),\ \mathit{free}(t) \subseteq \mathcal{V},$
$P_{17}[\Phi](\Phi'),\ \underline{P_{26}[\Lambda_*](t,i)},\ \underline{P_{39}[\Lambda_*](t)},\ \underline{P_{42}[\Lambda_*](t,i)},\ S_4[\mathcal{S},*.\mathit{find}](t)$

$\underline{P_{26}[\Lambda_*](t,i)}$: Follows from 116 by if-condition, and from 126 by $\mathcal{T}(t) \neq i$, $t \in \Lambda_{\mathcal{T}(t)}$, and $t \in \Lambda_i$.

$\underline{P_{39}[\Lambda_*](t)}$**:** Follows from 116 by $P_{=}(all)$ and follows trivially from 126.

$\underline{P_{42}[\Lambda_*](t,i)}$**:** Follows from 116 by $P_{=}(all)$ and follows trivially from 126.

Line 130:

128. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*),\ fr(t),\ P_{23}(t),\ free(t)\subseteq\mathcal{V},$
$P_{17}[\Phi](\Phi'),\ P_{26}[\Lambda_*](t,i),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.find](t)$
129. `IF HasFind(`t`) THEN BEGIN`
130. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*),\ fr(t),\ P_{17}[\Phi](\Phi'),\ hf(t),$
$\underline{P_{24}[\Lambda_*](t,t)},\ P_{26}[\Lambda_*](t,i),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.find](t)$

$\underline{P_{24}[\Lambda_*](t,t)}$**:** Suppose $s\lhd d\trianglelefteq t$ and s occurs $\mathcal{T}(d)$-alien in d. Then since $hf(t)$, it follows by G_6 that $hf(d)$, so $d=d\in\mathcal{F}$. Thus, by G_9, $(s\in\Lambda_{\mathcal{T}(d)}\wedge s\in\Lambda_{\mathcal{T}(s)})$.

Line 132:

130. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*),\ fr(t),\ P_{17}[\Phi](\Phi'),\ hf(t),$
$P_{24}[\Lambda_*](t,t),\ P_{26}[\Lambda_*](t,i),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.find](t)$
131. `RETURN;`
132. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\underline{\mathcal{F},\mathcal{R}}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$
$P_{26}[\Lambda_*](t,i),\ \underline{P_{27}[*.find](t)},\ \underline{P_{28}[*.find](t)},\ \underline{t.find\equiv t},\ P_{39}[\Lambda_*](t),$
$P_{42}[\Lambda_*](t,i),\ \underline{\mathcal{M}\subseteq\mathcal{R}}$

$\underline{P_{\subseteq}(\mathcal{F},\mathcal{R})}$**:** By $P_{=}(\mathcal{S},*.find)$ at 130.

$\underline{P_{27}[*.find](t)}$**:** By $P_{=}(*.find)$ at 130.

$\underline{P_{28}[*.find](t)}$**:** By $P_{=}(*.find)$ at 130.

$\underline{t.find\equiv t}$**:** By $fr(t)$ and $hf(t)$ at 130.

$\underline{\mathcal{M}\subseteq\mathcal{R}}$**:** By $P_{=}(*.find)$ at 130.

Line 136:

134. G, $P_{=}(\mathcal{A},\mathcal{H},\mathcal{S},*.find)$, $P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*)$, $free(t)\subseteq\mathcal{V}$, $fr(t)$, $P_{17}[\Phi](\Phi')$, $P_{23}(t)$, $P_{26}[\Lambda_*](t,i)$, $\neg hf(t)$, $P_{39}[\Lambda_*](t)$, $P_{42}[\Lambda_*](t,i)$, $S_4[\mathcal{S},*.find](t)$

135. FOR k := 1 TO $Arity(t)$ DO BEGIN

136. G, $P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})$, $P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\underline{\mathcal{F},\mathcal{R}})$, $free(t)\subseteq\mathcal{V}$, $P_{17}[\Phi](\Phi')$, $P_{23}(t)$, $\neg hf(t)$, $P_{26}[\Lambda_*](t,i)$, $\underline{P_{27}[*.find](t)}$, $\underline{P_{28}[*.find](t)}$, $\underline{P_{29}[*.find](t,k)}$, $\underline{P_{30}[*.find](t,Arity(t)+1)}$, $\underline{P_{32}[\Lambda_*](t,k)}$, $P_{39}[\Lambda_*](t)$, $P_{42}[\Lambda_*](t,i)$, $S_4[\mathcal{S},*.find](t)$, $\underline{\mathcal{M}\subseteq\mathcal{R}}$

...

138. G, $P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})$, $P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R})$, $free(t)\subseteq\mathcal{V}$, $P_{17}[\Phi](\Phi')$, $P_{23}(t)$, $\neg hf(t)$, $P_{26}[\Lambda_*](t,i)$, $P_{27}[*.find](t)$, $P_{28}[*.find](t)$, $P_{29}[*.find](t,k+1)$, $P_{30}[*.find](t,Arity(t)+1)$, $P_{32}[\Lambda_*](t,k+1)$, $P_{39}[\Lambda_*](t)$, $P_{42}[\Lambda_*](t,i)$, $S_4[\mathcal{S},*.find](t)$, $\mathcal{M}\subseteq\mathcal{R}$

139. END

$\underline{P_{\subseteq}(\mathcal{F},\mathcal{R})}$**:** From 134: by $P_{=}(\mathcal{S},*.find)$ at 134. From 138: trivial.

$\underline{P_{27}[*.find](t)}$**:** From 134: by $P_{=}(*.find)$ at 134. From 138: trivial.

$\underline{P_{28}[*.find](t)}$**:** From 134: by $P_{=}(*.find)$ at 134. From 138: trivial.

$\underline{P_{29}[*.find](t,k)}$**:** From 134: k=1. From 138: by $P_{29}[*.find](t,k+1)$ at 138.

$\underline{P_{30}[*.find](t,Arity(t)+1)}$**:** From 134: by $fr(t)$ and $\neg hf(t)$ at 134. From 138: trivial.

$\underline{P_{32}[\Lambda_*](t,k)}$**:** From 134: k=1. From 138: by $P_{32}[\Lambda_*](t,k+1)$ at 138.

$\underline{\mathcal{M}\subseteq\mathcal{R}}$**:** From 134: by $P_{=}(*.find)$ at 134. From 138: trivial.

Line 138:

136. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ \mathit{free}(t)\subseteq\mathcal{V},\ P_{17}[\Phi](\Phi'),\ P_{23}(t),$ $\neg \mathit{hf}(t),\ P_{26}[\Lambda_*](t,i),\ P_{27}[*.\mathit{find}](t),\ P_{28}[*.\mathit{find}](t),\ P_{29}[*.\mathit{find}](t,k),$ $P_{30}[*.\mathit{find}](t,\mathit{Arity}(t)+1),\ P_{32}[\Lambda_*](t,k),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$ $S_4[\mathcal{S},*.\mathit{find}](t),\ \mathcal{M}\subseteq\mathcal{R}$

137. `SetupTerm(`$t[k]$`,`$\mathcal{T}(t)$`);`

138. $\underline{G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ \mathit{free}(t)\subseteq\mathcal{V},\ P_{17}[\Phi](\Phi'),\ P_{23}(t),}$ $\underline{\neg \mathit{hf}(t),\ P_{26}[\Lambda_*](t,i),\ P_{27}[*.\mathit{find}](t),\ P_{28}[*.\mathit{find}](t),\ P_{29}[*.\mathit{find}](t,k+1),}$ $\underline{P_{30}[*.\mathit{find}](t,\mathit{Arity}(t)+1),\ P_{32}[\Lambda_*](t,k+1),\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),}$ $\underline{S_4[\mathcal{S},*.\mathit{find}](t),\ \mathcal{M}\subseteq\mathcal{R}}$

...

114. $G,\ \underline{\mathit{fr}(t)},\ \underline{P_{23}(t)},\ \mathit{free}(t)\subseteq\mathcal{V},\ \underline{S_4[\mathcal{S},*.\mathit{find}](t)}$

115. `SetupTerm(`t`,`i`)`

...

145. `END SetupTerm`

146. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$ $P_{26}[\Lambda_*](t,i),\ P_{27}[*.\mathit{find}](t),\ P_{28}[*.\mathit{find}](t),\ t.\mathit{find}\equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$ $\mathcal{M}\subseteq\mathcal{R}$

First we must show the preconditions of `SetupTerm` are met.

$\underline{\mathit{fr}(t[k])}$**:** By $P_{30}[*.\mathit{find}](t,\mathit{Arity}(t)+1)$ at 136.

$\underline{P_{23}(t[k])}$**:** By $P_{23}(t)$ at 136.

$\underline{S_4[\mathcal{S},*.\mathit{find}](t[k])}$**:** By $S_4[\mathcal{S},*.\mathit{find}](t)$ at 136.

Now we consider the properties at line 138.

$\underline{G}$**:** By G at 146.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})}$**:** By $P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})$ at 136 and 146.

$\underline{P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R})}$**:** By $P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R})$ at 136 and 146.

$\underline{\mathit{free}(t)\subseteq\mathcal{V}}$**:** By $\mathit{free}(t)\subseteq\mathcal{V}$ at 136 and $P_{\subseteq}(\mathcal{V})$ at 146.

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{17}[\Phi](\Phi')$ and $P_{\subseteq}(\Phi)$ at 136 and 146, Lemma A.4, and $P_{\subseteq}(\mathcal{V})$ at 136.

$\underline{P_{23}(t)}$**:** By $P_{23}(t)$ at 136.

$\underline{\neg hf(t)}$: By $\neg hf(t)$ at 136 and $P_{27}[*.find](t)$ at 146.

$\underline{P_{26}[\Lambda_*](t,i)}$: By $P_{26}[\Lambda_*](t,i)$ at 136 and $P_{\subseteq}(\Lambda_*)$ at 146.

$\underline{P_{27}[*.find](t)}$: By $P_{27}[*.find](t)$ at 136 and $P_{27}[*.find](t)$ at 146.

$\underline{P_{28}[*.find](t)}$: By $P_{28}[*.find](t)$ at 136 and $P_{27}[*.find](t)$ and $P_{28}[*.find](t)$ at 146.

$\underline{P_{29}[*.find](t,k+1)}$: By $P_{29}[*.find](t,k)$ at 136, $P_{27}[*.find](t)$ and $P_{28}[*.find](t)$ at 146, and $t.find \equiv t$ at 146.

$\underline{P_{30}[*.find](t,Arity(t)+1)}$: By $P_{30}[*.find](t,Arity(t)+1)$ at 136, and $P_{27}[*.find](t)$ and $P_{28}[*.find](t)$ at 146.

$\underline{P_{32}[\Lambda_*](t,k+1)}$: By $P_{32}[\Lambda_*](t,k)$ at 136 and $P_{\subseteq}(\Lambda_*)$ at 146, and $P_{24}[\Lambda_*](t,t)$ and $P_{26}[\Lambda_*](t,i)$ at 146.

$\underline{P_{39}[\Lambda_*](t)}$: By $P_{39}[\Lambda_*](t)$ at 136 and 146 and the fact that $t[k] \lhd t$.

$\underline{P_{42}[\Lambda_*](t,i)}$: Suppose $s \trianglelefteq t$, $s \in \Lambda_j$, and $j \neq \mathcal{T}(s)$. We must show that $s \in {\Lambda'}_j$, $s \equiv t \wedge j = i$, or s occurs j-alien in some sub-term of t. First note that if $s \in \Lambda_j$ at 136, then the required property follows by $P_{42}[\Lambda_*](t,i)$ at 136. Now, if $s \not\trianglelefteq t[k]$, then by $P_{39}[\Lambda_*](t)$ at 146, $s \in \Lambda_j$ at 136. Finally, we consider the case where $s \trianglelefteq t[k]$. In this case, by $P_{42}[\Lambda_*](t,i)$ at 146, we have $s \in \Lambda_{j,136}$ (i.e. s in Λ_j at line 136 in which case the required property follows as shown above), $s \equiv t[k] \wedge j = \mathcal{T}(t)$, or s occurs j-alien in some sub-term of $t[k]$ (and thus, clearly in some sub-term of t). The only non-trivial case is the middle one: $s \equiv t[k] \wedge j = \mathcal{T}(t)$. But we know that $j \neq \mathcal{T}(s)$, so $\mathcal{T}(t[k]) \neq \mathcal{T}(t)$, and thus $t[k]$ is j-alien in t. Since $s \equiv t[k]$, it follows that s occurs j-alien in some sub-term of t (in this case, the sub-term is t itself).

$\underline{S_4[\mathcal{S},*.find](t)}$: Suppose $s \trianglelefteq t$. We must show $\neg hf(t) \rightarrow (t \equiv canon_\chi(\mathcal{S}(t)))$. If $s \trianglelefteq t[k]$, then $hf(s)$ by $t.find \equiv t$ at 146 and G_6. Otherwise, $s.find \equiv s.find_{136}$ by $P_{27}[*.find](t)$ at 146, so $t \equiv canon_\chi(\mathcal{S}(t))$ by $S_4[\mathcal{S},*.find](t)$ at 136 and $P_{=}(\mathcal{S})$ at 146.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$**:** By $\mathcal{M} \subseteq \mathcal{R}$ at 136 and 146 and $P_{\subseteq}(\mathcal{R})$ at 146.

Line 140:

134. G, $P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S}, *.find)$, $P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*)$, $free(t) \subseteq \mathcal{V}$, $fr(t)$, $P_{17}[\Phi](\Phi')$, $P_{23}(t)$, $P_{26}[\Lambda_*](t, i)$, $\neg hf(t)$, $P_{39}[\Lambda_*](t)$, $P_{42}[\Lambda_*](t, i)$, $S_4[\mathcal{S}, *.find](t)$

135. `FOR` k `:= 1 TO` $Arity(t)$ `DO BEGIN`

...

138. G, $P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S})$, $P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*, \mathcal{F}, \mathcal{R})$, $free(t) \subseteq \mathcal{V}$, $P_{17}[\Phi](\Phi')$, $P_{23}(t)$, $\neg hf(t)$, $P_{26}[\Lambda_*](t, i)$, $P_{27}[*.find](t)$, $P_{28}[*.find](t)$, $P_{29}[*.find](t, k+1)$, $P_{30}[*.find](t, Arity(t)+1)$, $P_{32}[\Lambda_*](t, k+1)$, $P_{39}[\Lambda_*](t)$, $P_{42}[\Lambda_*](t, i)$, $S_4[\mathcal{S}, *.find](t)$, $\mathcal{M} \subseteq \mathcal{R}$

139. `END`

140. G, $P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{S})$, $P_{\subseteq}(\Phi, \mathcal{V}, \mathcal{N}, \Lambda_*, \mathcal{F}, \mathcal{R})$, $free(t) \subseteq \mathcal{V}$, $P_{17}[\Phi](\Phi')$, $P_{23}(t)$, $\underline{P_{24}[\Lambda_*](t, t)}$, $P_{26}[\Lambda_*](t, i)$, $\neg hf(t)$, $\underline{P_{27}[*.find](t)}$, $\underline{P_{28}[*.find](t)}$, $\underline{P_{31}[*.find](t)}$, $P_{39}[\Lambda_*](t)$, $P_{42}[\Lambda_*](t, i)$, $S_4[\mathcal{S}, *.find](t)$, $\mathcal{M} \subseteq \mathcal{R}$

$\underline{P_{24}[\Lambda_*](t, t)}$**:** From 134: trivial since t has no sub-terms. From 138: suppose $s \lhd d \trianglelefteq t$ and s occurs $\mathcal{T}(d)$-alien in d. We must show $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$. If $d \lhd t$, then for some child c of t, $d \trianglelefteq c$, so $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$ by $P_{24}[\Lambda_*](c, c)$ which holds by $P_{32}[\Lambda_*](t, k+1)$ at 138 and the end-of-loop condition. If $d \equiv t$ and s is not a child of t, then there exists a child c of t such that $s \lhd c$ and s occurs $\mathcal{T}(d)$-alien in c, so by the same argument as above, $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$. Finally, if $d \equiv t$ and s is a child of t, then $s \in \Lambda_{\mathcal{T}(s)} \wedge s \in \Lambda_{\mathcal{T}(d)}$ by $P_{26}[\Lambda_*](s, \mathcal{T}(d))$ which holds by $P_{32}[\Lambda_*](t, k+1)$ at 138 and the end-of-loop condition.

$\underline{P_{27}[*.find](t)}$**:** From 134: By $P_{=}(*.find)$ at 134. From 138: trivial.

$\underline{P_{28}[*.find](t)}$**:** From 134: By $P_{=}(*.find)$ at 134. From 138: trivial.

$\underline{P_{31}[*.find](t)}$**:** From 134: trivial since t has no children. From 138: by $P_{29}[*.find](t, k+1)$ at 138 and the end-of-loop condition.

Line 142:

140. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ \mathit{free}(t)\subseteq\mathcal{V},\ P_{17}[\Phi](\Phi'),\ P_{23}(t),$ $P_{24}[\Lambda_*](t,t),\ P_{26}[\Lambda_*](t,i),\ \neg\mathit{hf}(t),\ P_{27}[*.\mathit{find}](t),\ P_{28}[*.\mathit{find}](t),\ P_{31}[*.\mathit{find}](t),$ $P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),\ S_4[\mathcal{S},*.\mathit{find}](t),\ \mathcal{M}\subseteq\mathcal{R}$

141. $t.\mathit{find}$ `:=` t`;` [G_{16}^{ok} `:= FALSE;`]

142. $\underline{G},\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\underline{\mathcal{F},\mathcal{R}}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$ $P_{26}[\Lambda_*](t,i),\ \underline{P_{27}[*.\mathit{find}](t)},\ \underline{P_{28}[*.\mathit{find}](t)},\ t.\mathit{find}\equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$ $\underline{\mathcal{M}\subseteq\mathcal{R}},\ \underline{S_{23}[*.\mathit{find},*.\mathit{notify}](t)}$

$\underline{G}$**:** We consider the global properties that depend on *find*. Note that by $\neg\mathit{hf}(t)$ at 140, it follows that $\mathit{HF}=\mathit{HF}_{140}\cup\{t\}$ and $\mathcal{F}=\mathcal{F}_{140}\cup\{t=t\}$.

$\underline{G_2}$**:** By G_2 at 140, $\mathcal{T}\cup\Phi\models\mathcal{F}_{140}$. Also, since $\mathcal{T}$ includes equality, and in particular reflexivity, $\mathcal{T}\cup\Phi\models\{t=t\}$. Thus, since $\mathcal{F}=\mathcal{F}_{140}\cup\{t=t\}$, $\mathcal{T}\cup\Phi\models\mathcal{F}$.

$\underline{G_5}$**:** By $\neg\mathit{hf}(t)$ and the execution of line 141, $\mathit{hf}(s)\equiv\mathit{hf}_{140}(s)$ and $\mathit{find}^*(s)=\mathit{find}^*_{140}(s)$ for all $s\not\equiv t$. But $\mathit{find}^*(t)\equiv t$, so G_5 holds at 142.

$\underline{G_6}$**:** By G_6 and $P_{31}[*.\mathit{find}](t)$ at 140.

$\underline{G_7}$**:** By G_7 and $P_{23}(t)$ at 140.

$\underline{G_8}$**:** By G_8 and $\mathit{free}(t)\subseteq\mathcal{V}$ at 140.

$\underline{G_9}$**:** Suppose r occurs j-alien in some sub-term s in $\Phi\cup\mathcal{F}$. We must show that $(r\in\Lambda_j\wedge r\in\Lambda_{\mathcal{T}(r)})$. If $s\trianglelefteq(\Phi\cup\mathcal{F}_{140})$, then $(r\in\Lambda_j\wedge r\in\Lambda_{\mathcal{T}(r)})$ by G_9 at 140. Otherwise, since $\mathcal{F}=\mathcal{F}_{140}\cup\{t=t\}$, it must be the case that $s\trianglelefteq t$. Thus, by $P_{24}[\Lambda_*](t,t)$ at 140, $(r\in\Lambda_j\wedge r\in\Lambda_{\mathcal{T}(r)})$.

$\underline{G_{11}}$**:** By G_{11} at 140 since $\mathcal{A}=\mathcal{A}_{140}$ and $\mathit{HF}_{140}\subseteq\mathit{HF}$.

$\underline{G_{14}}$**:** By G_{14} at 140 since $\mathcal{F}_{140}\subseteq\mathcal{F}$.

$\underline{G_{15}}$**:** By G_{15} at 140, $\mathcal{T}_\chi\cup\gamma_\chi(\mathcal{S})\models\gamma_\chi(\mathcal{F}_{140})$. Also, since $\mathcal{T}_\chi$ includes reflexivity, $\mathcal{T}_\chi\models\gamma_\chi(t=t)$. But $\mathcal{F}=\mathcal{F}_{140}\cup\{t=t\}$, so $\mathcal{T}_\chi\cup\gamma_\chi(\mathcal{S})\models\gamma_\chi(\mathcal{F})$.

$\underline{G_{16}}$**:** By definition, $\neg G_{16}^{ok}$ at 142.

$\underline{G_{17}}$**:** By G_{17} at 140 and since $\mathit{HF}_{140}\subseteq\mathit{HF}$.

$\underline{G_{18}}$**:** Notice that $fr_{140}(t)$ by $P_{31}[*.\mathit{find}](t)$ at 140 and $fr_{140}(t)$ at 142 since $t.\mathit{find} \equiv t$. It is not hard to see that as a result $fr(e) \leftrightarrow fr_{140}(e)$ for all expressions e. Thus G_{18} follows from G_{18} at 140.

$\underline{G_{19}}$**:** Suppose s is a χ-leaf. If $s \not\equiv t$, then by G_{19} at 140, $\mathit{hf}(s) \rightarrow (s.\mathit{find} \not\equiv s \leftrightarrow s \in \mathit{lhs}(\mathcal{S}))$ and $s \lhd \mathcal{S} \rightarrow \mathit{hf}(s)$ since $s.\mathit{find} \equiv s.\mathit{find}_{140}$ and $\mathcal{S} = \mathcal{S}_{140}$. Suppose $s \equiv t$. Clearly $\mathit{hf}(s)$ and $s.\mathit{find} \equiv s$. It remains to show that $s \notin \mathit{lhs}(\mathcal{S})$. Now, by G_{19} at 140, $s \lhd \mathcal{S} \rightarrow \mathit{hf}_{170}(s)$. But we know $\neg \mathit{hf}_{170}(s)$, so it follows that $s \not\lhd \mathcal{S}$, and thus $s \notin \mathit{lhs}(\mathcal{S})$.

$\underline{P_{\subseteq}(\mathcal{F}, \mathcal{R})}$**:** Given $P_{\subseteq}(\mathcal{F}, \mathcal{R})$ at 140, we must simply show $\mathcal{F}_{140} \subseteq \mathcal{F}$ and $\mathcal{R}_{140} \subseteq \mathcal{R}$. We already showed that $\mathcal{F} = \mathcal{F}_{140} \cup \{t = t\}$. To show that $\mathcal{R}_{140} \subseteq \mathcal{R}$, note that $\mathcal{S}_{140} = \mathcal{S}$ and $s.\mathit{find}_{140} \equiv s.\mathit{find}$ for $s \not\equiv t$. Thus, it only remains to show that $t \in \mathcal{R}$. But this follows by $\mathit{hf}(t)$ and $\neg \mathit{hf}(t)$ and $S_4[\mathcal{S}, *.\mathit{find}](t)$ at 140.

$\underline{P_{27}[*.\mathit{find}](t)}$**:** Follows by $P_{27}[*.\mathit{find}](t)$ at 140 since $s.\mathit{find}_{140} \equiv s.\mathit{find}$ for $s \not\equiv t$.

$\underline{P_{28}[*.\mathit{find}](t)}$**:** By $P_{28}[*.\mathit{find}](t)$ at 140, $s.\mathit{find}_{140} \equiv s.\mathit{find}$ for $s \not\equiv t$, and $t.\mathit{find} \equiv t$.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$**:** $\mathcal{M}$ differs from $\mathcal{M}_{140}$ only by the addition of t. We know $\mathcal{M}_{140} \subseteq \mathcal{R}_{140}$ and, as shown above, $\mathcal{R}_{140} \subseteq \mathcal{R}$. But we also know that $t \in \mathcal{R}$ (also shown above), so $\mathcal{M} \subseteq \mathcal{R}$.

$\underline{S_{23}[*.\mathit{find}, *.\mathit{notify}](t)}$**:** By G_{16} at 140 and $s.\mathit{find}_{140} \equiv s.\mathit{find}$ for $s \not\equiv t$.

Line 144:

142. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$
$P_{26}[\Lambda_*](t,i),\ P_{27}[*.\mathit{find}](t),\ P_{28}[*.\mathit{find}](t),\ t.\mathit{find} \equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$
$\mathcal{M} \subseteq \mathcal{R},\ S_{23}[*.\mathit{find}, *.\mathit{notify}](t)$

143. $\texttt{TheorySetup}_{\mathcal{T}(t)}\texttt{(}t\texttt{);}$ [G_{16}^{ok} := TRUE;]

144. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{S}),\ P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R}),\ P_{17}[\Phi](\Phi'),\ P_{24}[\Lambda_*](t,t),$
$P_{26}[\Lambda_*](t,i),\ P_{27}[*.\mathit{find}](t),\ P_{28}[*.\mathit{find}](t),\ t.\mathit{find} \equiv t,\ P_{39}[\Lambda_*](t),\ P_{42}[\Lambda_*](t,i),$
$\mathcal{M} \subseteq \mathcal{R}$

...

264. $G,\ \mathit{hf}(e)$

265. $\texttt{TheorySetup}_i(e)$

266. $G,\ P_{34}[\mathit{all}](i)$

...

366. $\texttt{END TheorySetup}_\chi$

367. $G,\ P_{34}[\mathit{all}](\chi),\ S_{35}[*.\mathit{notify}](e)$

The only nontrivial precondition is $\mathit{hf}(t)$ which is true by $t.\mathit{find} \equiv t$ at 142. Consider now the properties at 144. Recall that $P_{34}[\mathit{all}](i)$ at 266 implies $P_{=}(\Phi - \mathcal{B}_i, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, \Lambda_*, *.\mathit{find}),\ P_{\subseteq}(\mathcal{Q}, \mathcal{B}_i, \mathcal{V}, *.\mathit{notify}),\ \mathcal{T}_i \cup \gamma_i(\Phi_i' \models \exists\, \overline{w}.\, \mathcal{B}_i) \wedge \overline{w} \cap \mathcal{V}\,' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$, where $\overline{w} = \mathit{free}(\mathcal{B}_i) - \mathit{free}(\Phi_i')$, and $\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}')) \wedge \mathit{free}(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}$.

$\underline{G}$**:** All but G_{16} follow from G at 266. To show, G_{16}, suppose s is a compound χ-term and $\mathit{hf}(s)$. We must show $\forall\, c \in \delta_\chi(s).\ (\chi, s) \in c.\mathit{notify}$). If $s \not\equiv t$, then this follows by $S_{23}[*.\mathit{find}, *.\mathit{notify}](t)$ at 142 and $P_{=}(*.\mathit{find})$ and $P_{\subseteq}(*.\mathit{notify})$ at 266. If $s \equiv t$, then s must be a compound χ term, and thus $\texttt{TheorySetup}_\chi$ is called. Thus, by $S_{35}[*.\mathit{notify}](e)$ at 367, $\forall\, c \in \delta_\chi(s).\ (\chi, s) \in c.\mathit{notify}$).

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})}$**:** By $P_{=}(\mathcal{A},\mathcal{H},\mathcal{S})$ at 142 and 266.

$\underline{P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R})}$**:** By $P_{\subseteq}(\Phi,\mathcal{V},\mathcal{N},\Lambda_*,\mathcal{F},\mathcal{R})$ at 142 and $P_{=}(\mathcal{N},\mathcal{S},\Lambda_*,*.\mathit{find})$ and $P_{\subseteq}(\Phi,\mathcal{V})$ at 266.

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{17}[\Phi](\Phi')$ at 142, $\mathcal{T} \cup \Phi' \models \exists\, \overline{w}.\, \Phi_{142}$ and $\overline{w} \cap \mathcal{V}\,' = \emptyset$, where $\overline{w} = \mathit{free}(\Phi_{142}) - \mathit{free}(\Phi')$. Then, by $P_{34}[\mathit{all}](i)$ at 266, $\mathcal{T} \cup \Phi_{142} \models \exists\, \overline{x}.\Phi$ and $\overline{x} \cap \mathcal{V}_{142} = \emptyset$, where $\overline{x} = \mathit{free}(\Phi) - \mathit{free}(\Phi_{142})$. Thus, by $P_{\subseteq}(\Phi)$ at 142 and $\Phi = \Phi_{142} \cup \mathcal{B}_i$ at 266, we can use Lemma A.4 to get $\mathcal{T} \cup \Phi' \models \exists\, \overline{y}.\, \Phi$, where $\overline{y} = \overline{w} \cup \overline{x}\mathit{free}(\Phi) - \mathit{free}(\Phi')$. It follows easily (by $P_{\subseteq}(\mathcal{V})$ at 142) that $\overline{y} \cap \mathcal{V}\,' = \emptyset$.

$\underline{P_{24}[\Lambda_*](t,t)}$: By $P_{24}[\Lambda_*](t,t)$ at 142 and $P_=(\Lambda_*)$ at 266.

$\underline{P_{26}[\Lambda_*](t,i)}$: By $P_{26}[\Lambda_*](t,i)$ at 142 and $P_=(\Lambda_*)$ at 266.

$\underline{P_{27}[*.find](t)}$: By $P_{27}[*.find](t)$ at 142 and $P_=(*.find)$ at 266.

$\underline{P_{28}[*.find](t)}$: By $P_{28}[*.find](t)$ at 142 and $P_=(*.find)$ at 266.

$\underline{t.find \equiv t}$: By $t.find \equiv t$ at 142 and $P_=(*.find)$ at 266.

$\underline{P_{39}[\Lambda_*](t)}$: By $P_{39}[\Lambda_*](t)$ at 142 and $P_=(\Lambda_*)$ at 266.

$\underline{P_{42}[\Lambda_*](t,i)}$: By $P_{42}[\Lambda_*](t,i)$ at 142 and $P_=(\Lambda_*)$ at 266.

$\underline{\mathcal{M} \subseteq \mathcal{R}}$: By $\mathcal{M} \subseteq \mathcal{R}$ at 142 and $P_=(\mathcal{S}, *.find)$ at 266.

A.5.7 Simplify

Line 153:

```
151.      G, P_=(all), T ∪ Φ |= e' ≃ e, hf(e)
152.      RETURN Find(e);
153.      G, P_=(A, H, I, N, S, *.find), P_⊆(Φ, V), free(retval) ⊆ V, fr(retval),
          T ∪ Φ |= e' ≃ retval, P_17[Φ](Φ'), S_4[S, *.find](retval), S_5(retval)
...
237. G, hf(t)
238. Find(t)
...
250. END Find
251. G, P_=(all), retval.find ≡ retval, t ∼ retval
```

$\underline{G}$: By G at 251.

$\underline{P_=(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find)}$: By $P_=(all)$ at 151 and 251.

$\underline{P_\subseteq(\Phi, \mathcal{V})}$: By $P_=(all)$ at 151 and 251.

$\underline{free(retval) \subseteq \mathcal{V}}$: $retval = retval \in \mathcal{F}$ by $retval.find \equiv retval$ at 251, so $free(retval) \subseteq \mathcal{V}$ by G_8.

$\underline{fr(retval)}$: By $retval.find \equiv retval$ at 251.

$\underline{\mathcal{T} \cup \Phi \models e' \simeq \mathit{retval}}$: We know $\mathcal{T} \cup \Phi_{151} \models e' \simeq e$. Then, $\mathcal{F} \models e = \mathit{retval}$ by $t \sim \mathit{retval}$ at 251. But $\Phi_{151} = \Phi$ by $P_{=}(\Phi)$ at 251 and $\mathcal{T} \cup \Phi \models \mathcal{F}$ by G_2, so $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$ and thus $\mathcal{T} \cup \Phi \models e' \simeq \mathit{retval}$.

$\underline{P_{17}[\Phi](\Phi')}$: Trivial since $\Phi = \Phi'$ by $P_{=}(\Phi)$ at 151 and $P_{=}(\Phi)$ at 251.

$\underline{S_4[\mathcal{S}, *.\mathit{find}](\mathit{retval})}$: We know $\mathit{hf}(\mathit{retval})$ by $\mathit{retval}.\mathit{find} \equiv \mathit{retval}$ at 251, and thus $\mathit{hf}(t)$ for all $t \trianglelefteq \mathit{retval}$ by G_6. $S_4[\mathcal{S}, *.\mathit{find}](\mathit{retval})$ follows trivially.

$\underline{S_5(\mathit{retval})}$: As just shown, $\mathit{hf}(\mathit{retval})$, so by G_7, retval is a term, and thus $\mathit{Op}(\mathit{retval}) \neq$ '='.

Line 157:

155. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e' \simeq e,\ \neg \mathit{hf}(e)$
156. `FOR` k `:= 1 to` $\mathit{Arity}(e)$ `DO BEGIN`
157. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.\mathit{find}),\ P_{\subseteq}(\Phi, \mathcal{V}),\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e' \simeq e,$ $\underline{P_{17}[\Phi](\Phi')},\ \underline{P_{30}[*.\mathit{find}](e, k)},\ \underline{S_{24}[\mathcal{S}, *.\mathit{find}](e, k)}$

...

159. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.\mathit{find}),\ P_{\subseteq}(\Phi, \mathcal{V}),\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e' \simeq e,$ $P_{17}[\Phi](\Phi'),\ P_{30}[*.\mathit{find}](e, k+1),\ S_{24}[\mathcal{S}, *.\mathit{find}](e, k+1)$
160. `END`

It is easy to see that all properties are satisfied by the transition from 159. Thus it suffices to consider the transition from 155.

$\underline{P_{17}[\Phi](\Phi')}$: Trivial since $\Phi = \Phi'$ by $P_{=}(\mathit{all})$ at 155.

$\underline{P_{30}[*.\mathit{find}](e, k)}$: Trivial since k=1.

$\underline{S_{24}[\mathcal{S}, *.\mathit{find}](e, k)}$: Trivial since k=1.

Line 159:

```
157.    G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(e) ⊆ V, T ∪ Φ |= e' ≃ e,
        P_17[Φ](Φ'), P_30[*.find](e,k), S_24[S,*.find](e,k)
158.    e[k] := Simplify(e[k]);
159.    G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(e) ⊆ V, T ∪ Φ |= e' ≃ e,
        P_17[Φ](Φ'), P_30[*.find](e,k+1), S_24[S,*.find](e,k+1)
...
147. G, free(e) ⊆ V
148. Simplify(e)
...
164. END Simplify
165. G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(retval) ⊆ V, fr(retval),
     T ∪ Φ |= e' ≃ retval, P_17[Φ](Φ'), S_4[S,*.find](retval), S_5(retval)
```

$\underline{G}$**:** By G at 165.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find)}$**:** By $P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find)$ at 157 and 165.

$\underline{P_{\subseteq}(\Phi,\mathcal{V})}$**:** by $P_{\subseteq}(\Phi,\mathcal{V})$ at 157 and 165.

$\underline{free(e) \subseteq \mathcal{V}}$**:** By the execution of 158 and $free(retval) \subseteq \mathcal{V}$ at 165 together with $free(e) \subseteq \mathcal{V}$ at 157 and $P_{\subseteq}(\mathcal{V})$ at 165.

$\underline{\mathcal{T} \cup \Phi \models e' \simeq e}$**:** By the execution of 158, $\mathcal{T} \cup \Phi \models e' \simeq retval$ at 165, together with $\mathcal{T} \cup \Phi \models e' \simeq e$ at 158 and $P_{\subseteq}(\Phi)$ at 165 and the properties of substitution.

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{17}[\Phi](\Phi')$ and $P_{\subseteq}(\Phi)$ at 157 and 165, Lemma A.4, and $P_{\subseteq}(\mathcal{V})$ at 165.

$\underline{P_{30}[*.find](e,k+1)}$**:** By $P_{30}[*.find](e,k)$ at 157, $fr(retval)$ at 165, and $P_{=}(*.find)$ at 165.

$\underline{S_{24}[\mathcal{S},*.find](e,k+1)}$**:** By $S_{24}[\mathcal{S},*.find](e,k)$ at 157, $S_4[\mathcal{S},*.find](retval)$ at 165, and $P_{=}(\mathcal{S},*.find)$ at 165.

Line 161:

```
155.   G, P_=(all), free(e) ⊆ V, T ∪ Φ |= e' ≃ e, ¬hf(e)
156.   FOR k := 1 to Arity(e) DO BEGIN
...
159.     G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(e) ⊆ V, T ∪ Φ |= e' ≃ e,
         P_17[Φ](Φ'), P_30[*.find](e,k+1), S_24[S,*.find](e,k+1)
160.   END
161.   G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(e) ⊆ V, T ∪ Φ |= e' ≃ e,
       P_17[Φ](Φ'), P_30[*.find](e,Arity(e)+1), S_24[S,*.find](e,Arity(e)+1)
```

$\underline{P_{17}[\Phi](\Phi')}$**:** From 155: by $P_{=}(all)$ at 155. From 159: trivial.

$\underline{P_{30}[*.find](e, Arity(e)+1)}$**:** From 155: trivial since e has no children. From 159: by end-of-loop condition.

$\underline{S_{24}[\mathcal{S}, *.find](e, Arity(e)+1)}$**:** From 155: trivial since e has no children. From 159: by end-of-loop condition.

Line 163:

```
161.   G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(e) ⊆ V, T ∪ Φ |= e' ≃ e,
       P_17[Φ](Φ'), P_30[*.find](e,Arity(e)+1), S_24[S,*.find](e,Arity(e)+1)
162.   RETURN Rewrite(e);
163.   G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(retval) ⊆ V, fr(retval),
       T ∪ Φ |= e' ≃ retval, P_17[Φ](Φ'), S_4[S,*.find](retval), S_5(retval)
...
166. G, free(e) ⊆ V, fr(e) ∨ hf(e), S_25[S,*.find](e)
167. Rewrite(e)
...
185. END Rewrite
186. G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(retval) ⊆ V, fr(retval),
     T ∪ Φ |= e ≃ retval, P_17[Φ](Φ'), S_4[S,*.find](retval), S_5(retval)
```

First we consider the preconditions of `Rewrite`.

$\underline{fr(e) \vee hf(e)}$**:** Suppose $\neg hf(e)$. By $P_{30}[*.find](e, Arity(e)+1)$ at 161, $fr(c)$ for all children c of e. Thus $fr(e)$.

$\underline{S_{25}[\mathcal{S}, *.find](e)}$**:** By $S_{24}[\mathcal{S}, *.find](e, Arity(e)+1)$ at 161.

Now we consider the properties at line 163.

$\underline{G}$: By G at 186.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find})}$: By $P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find})$ at 161 and 186.

$\underline{P_{\subseteq}(\Phi,\mathcal{V})}$: By $P_{\subseteq}(\Phi,\mathcal{V})$ at 161 and 186.

$\underline{\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}}$: By $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$ at 186.

$\underline{\mathit{fr}(\mathit{retval})}$: By $\mathit{fr}(\mathit{retval})$ at 186.

$\underline{\mathcal{T} \cup \Phi \models e' \simeq \mathit{retval}}$: By $\mathcal{T} \cup \Phi \models e' \simeq \mathit{retval}$ at 161, $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$ at 186, and $P_{\subseteq}(\Phi)$ at 186.

$\underline{P_{17}[\Phi](\Phi')}$: By $P_{17}[\Phi](\Phi')$ and $P_{\subseteq}(\Phi)$ at 161 and 186, Lemma A.4, and $P_{\subseteq}(\mathcal{V})$ at 186.

$\underline{S_4[\mathcal{S},*.\mathit{find}](\mathit{retval})}$: By $S_4[\mathcal{S},*.\mathit{find}](\mathit{retval})$ at 186.

$\underline{S_5(\mathit{retval})}$: By $S_5(\mathit{retval})$ at 186.

A.5.8 Rewrite

Line 172:

```
170.      G, P_=(all), free(e) ⊆ V, hf(e)
171.      RETURN Find(e);
172.      G, P_=(A,H,I,N,S,*.find), P_⊆(Φ,V), free(retval) ⊆ V, fr(retval),
          T ∪ Φ |= e ≃ retval, P_17[Φ](Φ'), S_4[S,*.find](retval), S_5(retval)
...
237. G, hf(t)
238. Find(t)
...
250. END Find
251. G, P_=(all), retval.find ≡ retval, t ~ retval
```

The preconditions of `Find` are satisfied trivially, so we just consider the properties at 172.

$\underline{G}$: By G at 251.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find})}$: By $P_{=}(\mathit{all})$ at 170 and 251.

$\underline{P_{\subseteq}(\Phi,\mathcal{V})}$: By $P_{=}(\mathit{all})$ at 170 and 251.

$\underline{\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}}$: $\mathit{retval} = \mathit{retval} \in \mathcal{F}$ by $\mathit{retval}.\mathit{find} \equiv \mathit{retval}$ at 251, so $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$ by G_8.

$\underline{\mathit{fr}(\mathit{retval})}$: By $\mathit{retval}.\mathit{find} \equiv \mathit{retval}$ at 251.

$\underline{\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}}$: $\mathcal{F} \models e = \mathit{retval}$ by $t \sim \mathit{retval}$ at 251 and $\mathcal{T} \cup \Phi \models \mathcal{F}$ by G_2, so $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$.

$\underline{P_{17}[\Phi](\Phi')}$: Trivial since $\Phi = \Phi'$ by $P_{=}(\mathit{all})$ at 170 and $P_{=}(\mathit{all})$ at 251.

$\underline{S_4[\mathcal{S},*.\mathit{find}](\mathit{retval})}$: We know $\mathit{hf}(\mathit{retval})$ by $\mathit{retval}.\mathit{find} \equiv \mathit{retval}$ at 251, and thus $\mathit{hf}(t)$ for all $t \trianglelefteq \mathit{retval}$ by G_6. $S_4[\mathcal{S},*.\mathit{find}](\mathit{retval})$ follows trivially.

$\underline{S_5(\mathit{retval})}$: As just shown, $\mathit{hf}(\mathit{retval})$, so by G_7, retval is a term, and thus $\mathit{Op}(\mathit{retval}) \neq$ '='.

Line 176:

```
174.   G, P=(all), free(e) ⊆ V, fr(e), S25[S, *.find](e)
175.   e* := OpRewrite(e);
176.   G, P=(A, H, I, N, S, *.find), P⊆(Φ, V), free(e*) ⊆ V, T ∪ Φ |= e ≃ e*,
       fr(e*), P17[Φ](Φ'), S26[S, *.find](e, e*), S27(e, e*)
...
187. G, free(e) ⊆ V, fr(e), S25[S, *.find](e)
188. OpRewrite(e)
...
210. END OpRewrite
211. G, P=(A, H, I, N, S, *.find), P⊆(Φ, V), free(retval) ⊆ V, fr(retval),
     T ∪ Φ |= e ≃ retval, P17[Φ](Φ'), S26[S, *.find](e, retval), S27(e, retval)
```

The preconditions of `OpRewrite` are satisfied trivially, so we just consider the properties at 176.

$\underline{G}$: By G at 211.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find})}$: By $P_{=}(\mathit{all})$ at 174 and $P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find})$ at 211.

$\underline{P_{\subseteq}(\Phi,\mathcal{V})}$: By $P_{=}(\mathit{all})$ at 174 and $P_{\subseteq}(\Phi,\mathcal{V})$ at 211.

$\underline{\mathit{free}(e^*) \subseteq \mathcal{V}}$: By $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$ at 211.

$\underline{\mathcal{T} \cup \Phi \models e \simeq e^*}$: By $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$ at 211.

$\underline{\mathit{fr}(e^*)}$: By $\mathit{fr}(\mathit{retval})$ at 211.

$\underline{P_{17}[\Phi](\Phi')}$: By $\Phi' = \Phi_{174}$ and $\mathcal{V}\,' = \mathcal{V}_{174}$ at 174 (by $P_{=}(\mathit{all})$) and $P_{17}[\Phi](\Phi')$ at 211.

$\underline{S_{26}[\mathcal{S},*.\mathit{find}](e,e^*)}$: By $S_{26}[\mathcal{S},*.\mathit{find}](e,\mathit{retval})$ at 211.

$\underline{S_{27}(e,e^*)}$: By $S_{27}(e,\mathit{retval})$ at 211.

Line 178:

176. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,$
$\mathit{fr}(e^*),\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S},*.\mathit{find}](e,e^*),\ S_{27}(e,e^*)$
177. `IF` $e \not\equiv e^*$ `THEN BEGIN`
178. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,$
$\mathit{fr}(e^*),\ P_{17}[\Phi](\Phi'),\ \underline{S_{25}[\mathcal{S},*.\mathit{find}](e)}$

$\underline{S_{25}[\mathcal{S},*.\mathit{find}](e)}$: By $S_{26}[\mathcal{S},*.\mathit{find}](e,e^*)$ at 176 and the if-condition.

Line 180:

178. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,$
$\mathit{fr}(e^*),\ P_{17}[\Phi](\Phi'),\ S_{25}[\mathcal{S},*.\mathit{find}](e)$
179. e^* `:= Rewrite(`e^*`);`
180. $\underline{G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,}$
$\underline{\mathit{fr}(e^*),\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S},*.\mathit{find}](e^*),\ S_5(e^*)}$

...

166. $G,\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathit{fr}(e) \vee \mathit{hf}(e),\ S_{25}[\mathcal{S},*.\mathit{find}](e)$
167. `Rewrite(`e`)`

...

185. `END Rewrite`
186. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(\mathit{retval}) \subseteq \mathcal{V},\ \mathit{fr}(\mathit{retval}),$
$\mathcal{T} \cup \Phi \models e \simeq \mathit{retval},\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S},*.\mathit{find}](\mathit{retval}),\ S_5(\mathit{retval})$

The preconditions of `Rewrite` are satisfied trivially, so we just consider the properties

at 180.

$\underline{G}$**:** By G at 186.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find})}$**:** By $P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find})$ at 178 and 186.

$\underline{P_{\subseteq}(\Phi,\mathcal{V})}$**:** By $P_{\subseteq}(\Phi,\mathcal{V})$ at 178 and 186.

$\underline{\mathit{free}(e^*) \subseteq \mathcal{V}}$**:** By $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$ at 186.

$\underline{\mathcal{T} \cup \Phi \models e \simeq e^*}$**:** By $\mathcal{T} \cup \Phi \models e \simeq e^*$ at 178, $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$ at 186, and $P_{\subseteq}(\Phi)$ at 186.

$\underline{\mathit{fr}(e^*)}$**:** By $\mathit{fr}(\mathit{retval})$ at 186.

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{17}[\Phi](\Phi')$ and $P_{\subseteq}(\Phi)$ at 178 and 186, Lemma A.4, and $P_{\subseteq}(\mathcal{V})$ at 186.

$\underline{S_4[\mathcal{S},*.\mathit{find}](e^*)}$**:** By $S_4[\mathcal{S},*.\mathit{find}](\mathit{retval})$ at 186.

$\underline{S_5(e^*)}$**:** By $S_5(\mathit{retval})$ at 186.

Line 182:

176. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,$
$\mathit{fr}(e^*),\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S},*.\mathit{find}](e,e^*),\ S_{27}(e,e^*)$
177. `IF` $e \not\equiv e^*$ `THEN BEGIN`
...
180. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,$
$\mathit{fr}(e^*),\ P_{17}[\Phi](\Phi'),\ S_4[\mathcal{S},*.\mathit{find}](e^*),\ S_5(e^*)$
181. `END`
182. $G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.\mathit{find}),\ P_{\subseteq}(\Phi,\mathcal{V}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq e^*,$
$\mathit{fr}(e^*),\ P_{17}[\Phi](\Phi'),\ \underline{S_4[\mathcal{S},*.\mathit{find}](e^*)},\ \underline{S_5(e^*)}$

The transition from 180 is trivial. Thus we consider only the transition from 176.

$\underline{S_4[\mathcal{S},*.\mathit{find}](e^*)}$**:** By $S_{26}[\mathcal{S},*.\mathit{find}](e,e^*)$ at 176 and the if-condition.

$\underline{S_5(e^*)}$**:** By $S_{27}(e,e^*)$ at 176 and the if-condition.

A.5.9 OpRewrite

Line 191:

189. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_{25}[\mathcal{S}, *.find](e)$
190. `IF` $Op(e) =$ '$\neg$' `THEN BEGIN`
191. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $\underline{S_4[\mathcal{S}, *.find](e)},\ \underline{S_5(e)}$

$\underline{S_4[\mathcal{S}, *.find](e)}$**:** By $S_{25}[\mathcal{S}, *.find](e)$ at 189 and the fact that e itself is not a term (since $Op(e) =$ '$\neg$').

$\underline{S_5(e)}$**:** Trivial since $Op(e) \neq$ '='.

Line 193:

191. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $S_4[\mathcal{S}, *.find](e),\ S_5(e)$
192. `RETURN RewriteNegation(`e`);`
193. $\underline{G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),}$
$\underline{\mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S}, *.find](e, retval),\ S_{27}(e, retval)}$

...

212. $G,\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) =$ '$\neg$', $S_4[\mathcal{S}, *.find](e)$
213. `RewriteNegation(`e`)`

...

235. `END RewriteNegation`
236. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),$
$S_4[\mathcal{S}, *.find](retval)$

The preconditions of `RewriteNegation` are satisfied trivially, so we just consider the properties at 193.

$\underline{G}$**:** By G at 236.

$\underline{P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find)}$**:** By $P_{=}(all)$ at 191 and 236.

$\underline{P_{\subseteq}(\Phi, \mathcal{V})}$**:** By $P_{=}(all)$ at 191 and 236.

$\underline{free(retval) \subseteq \mathcal{V}}$**:** By $free(retval) \subseteq \mathcal{V}$ at 236.

$\underline{fr(retval)}$**:** By $fr(retval)$ at 236.

$\underline{\mathcal{T} \cup \Phi \models e \simeq retval}$**:** By $\mathcal{T} \cup \Phi \models e \simeq retval$ at 236.

$\underline{P_{17}[\Phi](\Phi')}$: By $P_{=}(all)$ at 191 and 236.

$\underline{S_{26}[\mathcal{S}, *.find](e, retval)}$: By $S_4[\mathcal{S}, *.find](e)$ at 236.

$\underline{S_{27}(e, retval)}$: By $S_5(e)$ at 191.

Line 199:

```
197.    G, P_=(all), Op(e) = '=', e[1] ≡ e[2]
198.    RETURN true;
199.    G, P_=(A, H, I, N, S, *.find), P_⊆(Φ, V), free(retval) ⊆ V, fr(retval),
        T ∪ Φ |= e ≃ retval, P_17[Φ](Φ'), S_26[S, *.find](e, retval), S_27(e, retval)
```

$\underline{P_{17}[\Phi](\Phi')}$: By $P_{=}(all)$ at 197.

$\underline{S_{26}[\mathcal{S}, *.find](e, retval)}$: $S_4[\mathcal{S}, *.find](true)$ is trivially true since *true* has no sub-terms.

$\underline{S_{27}(e, retval)}$: $S_5(retval)$ is true since $Op(true) \neq$ '='.

Line 201:

```
195.  G, P_=(all), free(e) ⊆ V, fr(e), S_25[S, *.find](e)
196.  IF Op(e) = '=' AND e[1] ≡ e[2] THEN BEGIN
...
198.    RETURN true;
...
200.  END
201.  G, P_=(all), free(e) ⊆ V, fr(e), S_5(e), S_25[S, *.find](e)
```

$\underline{S_5(e)}$: $S_5(e)$ is the negation of the if-condition.

Line 203:

```
201.  G, P_=(all), free(e) ⊆ V, fr(e), S_5(e), S_25[S, *.find](e)
202.  IF e is a term or an atomic formula THEN BEGIN
203.    G, P_=(all), free(e) ⊆ V, fr(e), P_33(e), S_5(e), S_25[S, *.find](e)
```

$\underline{P_{33}(e)}$: By the if-condition.

Line 205:

203. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ \underline{P_{33}(e)},\ S_5(e),\ S_{25}[\mathcal{S}, *.find](e)$
204. `RETURN TheoryRewrite`$_{\mathcal{T}(e)}$`(e);`
205. $\underline{G,\ P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find),\ P_{\subseteq}(\Phi,\mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),\ \mathcal{T} \cup \Phi \models e \simeq retval,\ P_{17}[\Phi](\Phi'),\ S_{26}[\mathcal{S}, *.find](e, retval),\ S_{27}(e, retval)}$

...

261. $G,\ free(e) \subseteq \mathcal{V},\ fr(e),\ P_{33}(e),\ S_{25}[\mathcal{S}, *.find](e)$
262. `TheoryRewrite`$_i$`(e)`
263. $G,\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),\ free(retval) \subseteq \mathcal{V},\ P_{34}[all](i),\ S_{26}[\mathcal{S}, *.find](e, retval)$

The preconditions of `TheoryRewrite`$_i$ are satisfied trivially, so we consider the properties at 205. Recall that $P_{34}[all](i)$ at 263 implies $P_{=}(\Phi - \mathcal{B}_i, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, \Lambda_*, *.find)$, $P_{\subseteq}(\mathcal{Q}, \mathcal{B}_i, \mathcal{V}, *.notify)$, $\mathcal{T}_i \cup \gamma_i(\Phi_i' \models \exists \overline{w}.\, \mathcal{B}_i) \wedge \overline{w} \cap \mathcal{V}' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$, where $\overline{w} = free(\mathcal{B}_i) - free(\Phi_i')$, and $\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}')) \wedge free(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}$.

$\underline{G}$**:** By G at 263.

$\underline{P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find)}$**:** By $P_{=}(all)$ at 203 and $P_{=}(\mathcal{A},\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},*.find)$ at 263.

$\underline{P_{\subseteq}(\Phi,\mathcal{V})}$**:** By $P_{=}(all)$ at 203, $\Phi = \Phi_{203} \cup \mathcal{B}_i$, and $P_{\subseteq}(\mathcal{V})$ at 263.

$\underline{free(retval) \subseteq \mathcal{V}}$**:** By $free(retval) \subseteq \mathcal{V}$ at 263.

$\underline{fr(retval)}$**:** By $fr(retval)$ at 263.

$\underline{\mathcal{T} \cup \Phi \models e \simeq retval}$**:** By $\mathcal{T} \cup \Phi \models e \simeq retval$ at 263.

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{=}(all)$ at 203 and $P_{17}[\Phi](\Phi')$ at 263 (which, as we have shown before, follows from $P_{34}[all](i)$).

$\underline{S_{26}[\mathcal{S}, *.find](e, retval)}$**:** By $S_{26}[\mathcal{S}, *.find](e, retval)$ at 263.

$\underline{S_{27}(e, retval)}$**:** By $S_5(e)$ at 203.

Line 207:

201. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_5(e),\ S_{25}[\mathcal{S}, *.find](e)$
202. `IF` e `is a term or an atomic formula THEN BEGIN`
...
204. `RETURN ...`
...
206. `END`
207. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ \underline{S_4[\mathcal{S}, *.find](e)},\ S_5(e)$

$\underline{S_4[\mathcal{S}, *.find](e)}$**:** By $S_{25}[\mathcal{S}, *.find](e)$ at 201 and the fact that e itself is not a term by the if-condition.

Line 209:

207. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ S_4[\mathcal{S}, *.find](e),\ S_5(e)$
208. `RETURN` e`;`
209. $G,\ P_{=}(\mathcal{A}, \mathcal{H}, \mathcal{I}, \mathcal{N}, \mathcal{S}, *.find),\ P_{\subseteq}(\Phi, \mathcal{V}),\ free(retval) \subseteq \mathcal{V},\ fr(retval),$
$\mathcal{T} \cup \Phi \models e \simeq retval,\ \underline{P_{17}[\Phi](\Phi')},\ \underline{S_{26}[\mathcal{S}, *.find](e, retval)},\ \underline{S_{27}(e, retval)}$

$\underline{P_{17}[\Phi](\Phi')}$**:** By $P_{=}(all)$ at 207.

$\underline{S_{26}[\mathcal{S}, *.find](e, retval)}$**:** By $S_4[\mathcal{S}, *.find](e)$ at 207.

$\underline{S_{27}(e, retval)}$**:** By $S_5(e)$ at 207.

A.5.10 RewriteNegation

Line 218:

216. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) = \text{'}\neg\text{'},\ e[1] \equiv true$
217. `RETURN` $false$`;`
218. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),$
$\underline{S_4[\mathcal{S}, *.find](retval)}$

$\underline{S_4[\mathcal{S}, *.find](retval)}$**:** $S_4[\mathcal{S}, *.find](false)$ is trivially true since $false$ has no sub-terms.

Line 224:

222. $G,\ P_{=}(all),\ free(e) \subseteq \mathcal{V},\ fr(e),\ Op(e) = \text{'}\neg\text{'},\ e[1] \equiv false$
223. `RETURN` $true$`;`
224. $G,\ P_{=}(all),\ free(retval) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),$ $S_4[\mathcal{S}, *.find](retval)$

$S_4[\mathcal{S}, *.find](retval)$**:** $S_4[\mathcal{S}, *.find](true)$ is trivially true since *true* has no sub-terms.

A.5.11 Find

Line 247:

245. $G,\ P_{=}(all),\ hf(t)$
246. `RETURN Find(`$t.find$`);`
247. $G,\ P_{=}(all),\ retval.find \equiv retval,\ t \sim retval$

...

237. $G,\ hf(t)$
238. `Find(`t`)`

...

250. `END Find`
251. $G,\ P_{=}(all),\ retval.find \equiv retval,\ t \sim retval$

We first consider the preconditions of `Find`

$hf(\mathbf{t}.find)$**:** Follows from G_5 at 245.

We now consider the properties at 247.

G**:** By G at 251.

$P_{=}(all)$**:** By $P_{=}(all)$ at 245 and 251.

$retval.find \equiv retval$**:** By $retval.find \equiv retval$ at 251.

$t \sim retval$**:** By $hf(t)$, G_5, and the definition of $find^*$ and $\sim$, $t \sim_{245} t.find_{245}$. Then, by $P_{=}(all)$ at 251, $t \sim t.find$ Finally, by the `Find` postcondition, $t.find \sim retval$, so by the definition of $\sim$, $t \sim retval$.

A.5.12 Theory-Specific Code for a Nelson-Oppen Theory $\mathcal{T}_i$

TheoryAddSharedTerm

273. G, $\mathit{free}(e) \subseteq \mathcal{V}$
274. `TheoryAddSharedTerm`$_i$`(`e`)`
275. G, $\underline{P_{34}[all](i)}$

$\underline{P_{34}[all](i)}$: `TheoryAddSharedTerm`$_i$ does nothing, so $P_{=}(all)$ holds at 275, and thus, by Lemma A.5, so does $P_{34}[all](i)$.

TheoryAssert

276. G,
277. `TheoryAssert`$_i$`(`e`)`
278. G, $\underline{P_{34}[all](i)}$

$\underline{P_{34}[all](i)}$: `TheoryAssert`$_i$ does nothing, so $P_{=}(all)$ holds at 278, and thus, by Lemma A.5, so does $P_{34}[all](i)$.

TheoryCheckSat

Line 283:

281. G, $P_{=}(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, convex, $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$, $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$
282. `IF` $\neg \mathit{Sat}_i(\Phi_i \cup E_{\sim_i})$ `THEN BEGIN`
283. G, $P_{=}(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, convex, $\underline{\neg P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i)}$

$\underline{\neg P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i)}$: By the if-condition.

Line 285:

283. G, $P_{=}(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, convex, $\neg P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i)$
284. $\mathcal{I}$ `:= TRUE;`
285. $\underline{G}$, $P_{=}(all - \{\mathcal{I}\})$, $\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$, $\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](i)}$, $\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$, $\underline{S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]}$

$\underline{G}$: Only G_1 depends on $\mathcal{I}$. Since line 284 sets $\mathcal{I}$ to be `TRUE`, we must show that $\mathcal{T} \cup \mathcal{H} \models \mathit{false}$. By $\neg P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i)$ at 283, we have $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i}) \models$

false. It follows that $\mathcal{T} \cup \Phi \cup E_{\sim_i} \models \mathit{false}$. But by the definition of $E_{\sim_i}$, $\mathcal{F} \models E_{\sim_i}$, and $\mathcal{T} \cup \Phi \models \mathcal{F}$ by G_2 at 283, so $\mathcal{T} \cup \Phi \models \mathit{false}$. Finally, by G_3 at 283, if $\mathcal{T} \cup \mathcal{H}$ is satisfiable, then $\mathcal{T} \cup \mathcal{H} \cup \Phi$ is also satisfiable. Therefore, $\mathcal{T} \cup \mathcal{H}$ is not satisfiable.

$\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$**:** Trivial since $\mathcal{I} =$ `TRUE`;

$\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](i)}$**:** Trivial since $\mathcal{I} =$ `TRUE`;

$\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$**:** Trivial since $\mathcal{I} =$ `TRUE`;

$\underline{S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]}$**:** Trivial since $\mathcal{I} =$ `TRUE`;

Line 287:

281. $G,\ P_{=}(all),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \mathcal{Q} = \emptyset,\ \neg\mathcal{I},\ \mathit{convex},\ S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}],$ $S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$
282. `IF` $\neg \mathit{Sat}_i(\Phi_i \cup E_{\sim_i})$ `THEN BEGIN`
...
286. `END ELSE IF` $\neg \mathit{Sat}_i(\Phi_i \cup Ar_{\sim_i})$ `THEN BEGIN`
287. $G,\ P_{=}(all),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \mathcal{Q} = \emptyset,\ \neg\mathcal{I},\ \mathit{convex},\ \underline{\neg P_7[\Phi, \Lambda_*, *.\mathit{find}](i)},$ $\underline{P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i)},\ S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}],\ S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$

$\underline{\neg P_7[\Phi, \Lambda_*, *.\mathit{find}](i)}$**:** By the if-condition on line 286.

$\underline{P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i)}$**:** By the if-condition on line 282.

Line 289:

287. $G,\ P_{=}(all),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \mathcal{Q} = \emptyset,\ \neg\mathcal{I},\ \mathit{convex},\ \neg P_7[\Phi, \Lambda_*, *.\mathit{find}](i),$ $P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i),\ S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}],\ S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$
288. `Choose` $\Delta \subseteq D_{\sim_i}$ `such that` $\neg \mathit{Sat}_i(\Phi_i \cup E_{\sim_i} \cup \Delta)$`;`
289. $G,\ P_{=}(all),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \mathcal{Q} = \emptyset,\ \neg\mathcal{I},\ \mathit{convex},\ \neg P_7[\Phi, \Lambda_*, *.\mathit{find}](i),$ $P_{36}[\Phi, \Lambda_*, *.\mathit{find}](i),\ \underline{P_{37}[\Phi, \Lambda_*, *.\mathit{find}](i, \neg\Delta)},\ S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}],\ S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]$

A note on line 288: It is always possible to choose an appropriate set for Δ. In particular, $D_{\sim_i}$ always works since $E_{\sim_i} \cup D_{\sim_i} = Ar_{\sim_i}$ by definition and $\neg \mathit{Sat}_i(\Phi_i \cup Ar_{\sim_i})$ by $\neg P_7[\Phi, \Lambda_*, *.\mathit{find}](i)$ at 287.

$\underline{P_{37}[\Phi, \Lambda_*, *.find](i, \neg\Delta)}$**:** By execution of line 288, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i} \cup \Delta)$ is not satisfiable, so any model and variable assignment satisfying $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i})$ must not satisfy $\gamma_i(\Delta)$. Thus, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i}) \models \gamma_i(\neg\Delta)$.

Line 291:

289. G, $P_=(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, *convex*, $\neg P_7[\Phi, \Lambda_*, *.find](i)$, $P_{36}[\Phi, \Lambda_*, *.find](i)$, $P_{37}[\Phi, \Lambda_*, *.find](i, \neg\Delta)$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

290. $\mathcal{Q}$:= $\{\neg\Delta\}$;

291. $\underline{G}$, $P_=(all - \{\mathcal{I}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)}$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

$\underline{G}$**:** We must consider only those global properties which depend on $\mathcal{Q}$.

$\underline{G_4}$**:** By $P_{37}[\Phi, \Lambda_*, *.find](i, \neg\Delta)$ at 289, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup E_{\sim_i}) \models \gamma_i(\neg\Delta)$, so $\mathcal{T} \cup \Phi \cup E_{\sim_i} \models \neg\Delta$. But by the definition of $E_{\sim_i}$, $\mathcal{F} \models E_{\sim_i}$, and $\mathcal{T} \cup \Phi \models \mathcal{F}$ by G_2 at 289, so $\mathcal{T} \cup \Phi \models \neg\Delta$. G_4 follows by G_4 at 289 and the execution of line 290.

$\underline{G_8}$**:** By G_8 at 289, $\mathit{free}(\Lambda) \subseteq \mathcal{V}$. But $\mathit{free}(\Delta) \subseteq \mathit{free}(\Lambda)$, so $\mathit{free}(\mathcal{Q}) \subseteq \mathcal{V}$. The rest follows by G_8 at 289.

$\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)}$**:** Trivial, since $\mathcal{Q} \neq \emptyset$.

Line 293:

281. G, $P_=(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, *convex*, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

282. IF $\neg \mathit{Sat}_i(\Phi_i \cup E_{\sim_i})$ THEN BEGIN

...

285. G, $P_=(all - \{\mathcal{I}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

286. END ELSE IF $\neg \mathit{Sat}_i(\Phi_i \cup \mathit{Ar}_{\sim_i})$ THEN BEGIN

...

291. G, $P_=(all - \{\mathcal{I}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

292. END

293. G, $P_=(all - \{\mathcal{I}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](i)}$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

$\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](i)}$**:** The transitions from 285 and 291 are trivial. But if neither if-branch is taken, then by the if-condition of 286, $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ is satisfiable, from which $P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](i)$ follows easily.

TheoryRewrite

296. G, $\mathit{free}(e) \subseteq \mathcal{V}$, $\mathit{fr}(e)$, $P_{33}(e)$, $S_{25}[\mathcal{S}, *.\mathit{find}](e)$
297. `TheoryRewrite`$_i$`(`e`)`
298. `  RETURN` e`;`
299. G, $\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}$, $\mathit{fr}(\mathit{retval})$, $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$, $\underline{P_{34}[\mathit{all}](i)}$, $\underline{S_{26}[\mathcal{S}, *.\mathit{find}](e, \mathit{retval})}$

$\underline{P_{34}[\mathit{all}](i)}$**:** Since no state variables change, $P_=(\mathit{all})$ holds at 299, so by Lemma A.5, so does $P_{34}[\mathit{all}](i)$.

$\underline{S_{26}[\mathcal{S}, *.\mathit{find}](e, \mathit{retval})}$**:** Since $e \equiv \mathit{retval}$, we must show $S_4[\mathcal{S}, *.\mathit{find}](e)$. Suppose $t \trianglelefteq e$. If $t \not\equiv e$, then by $S_{25}[\mathcal{S}, *.\mathit{find}](e)$ at 296, $t \equiv \mathit{canon}_\chi(\mathcal{S}(t))$. Suppose $t \equiv e$ and $\neg \mathit{hf}(t)$, We must show $t \equiv \mathit{canon}_\chi(\mathcal{S}(t))$. First note that since $\mathcal{T}(e) = i$, and $\mathcal{T}_i$ is a Nelson-Oppen theory, t is not a χ-term, and thus t is a χ-leaf. Thus, by G_{19}, $t \not\triangleleft \mathcal{S}$, so $\mathcal{S}(t) \equiv t$. But by property 4 of *canon*, $\mathit{canon}_\chi(t) \equiv t$. Thus, $t \equiv \mathit{canon}_\chi(\mathcal{S}(t))$.

TheorySetup

300. G, $\mathit{hf}(e)$
301. `TheorySetup`$_i$`(`e`)`
302. G, $\underline{P_{34}[\mathit{all}](i)}$

$\underline{P_{34}[\mathit{all}](i)}$**:** `TheorySetup`$_i$ does nothing, so $P_=(\mathit{all})$ holds at 302, and thus, by Lemma A.5, so does $P_{34}[\mathit{all}](i)$.

TheorySolve

303. G, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e)$, $\mathit{free}(e)\subseteq\mathcal{V}$, $\mathit{fr}(e)$, $P_{11}(e)$, $\mathit{Op}(e)=\text{`='}$, $e[1]\not\equiv e[2]$, $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $S_4[\mathcal{S},*.\mathit{find}](e)$
304. `TheorySolve(`e`)`
305. `  RETURN` $\{e\}$;
306. G, $P_{=}(\mathcal{H},\mathcal{I},\mathcal{N},*.\mathit{find})$, $P_{\subseteq}(\Phi)$, $\mathit{free}(\mathit{retval})\subseteq\mathcal{V}$, $\underline{P_{14}[*.\mathit{find}](\mathit{retval})}$, $\underline{P_{17}[\Phi](\Phi')}$, $\underline{P_{35}[\Phi](e,\mathit{retval})}$, $S_2[\mathcal{I},\mathcal{S},*.\mathit{find}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $\underline{S_7[\mathcal{S},*.\mathit{find}](\mathit{retval})}$

Note that this (the default) implementation of `TheorySolve` is only used if there is *no* Shostak theory, since otherwise, the Shostak theory provides the implementation for `TheorySolve`.

$\underline{P_{14}[*.\mathit{find}](\mathit{retval})}$**:** We know that $\mathit{retval}=\{e\}$. $\forall\, e\in\mathit{retval}.\,\mathit{fr}(e)$ follows by $\mathit{fr}(e)$ at 303, and $P_{13}(\mathit{retval})$ follows from $P_{11}(e)$ and $\mathit{Op}(e)=\text{`='}$ at 303.

$\underline{P_{17}[\Phi](\Phi')}$**:** Follows easily since $\Phi'=\Phi$.

$\underline{P_{35}[\Phi](e,\mathit{retval})}$**:** Trivial since $\mathit{retval}=\{e\}$.

$\underline{S_7[\mathcal{S},*.\mathit{find}](\mathit{retval})}$**:** Given that $\mathit{retval}=\{e\}$, it is easy to see that $S_4[\mathcal{S},*.\mathit{find}](\mathit{retval})$ follows from $S_4[\mathcal{S},*.\mathit{find}](e)$ at 303. Then, since there is no Shostak theory, $e[1]$ and $e[2]$ are not χ-terms, and thus, since $e[1]\not\equiv e[2]$, $\{e\}$ is in χ-solved form (strictly speaking, this property isn't even needed if there is no Shostak theory).

TheoryUpdate

307. G, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $S_{28}[*.\mathit{find}](i,d)$
308. `TheoryUpdate`$_i$`(`e`,`d`)`
309. G, $P_{\subseteq}(\mathcal{F},\mathit{lhs}(\mathcal{S}))$, $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$, $\underline{\mathcal{I}\vee(\mathcal{M}\subseteq\mathcal{R})}$, $\mathcal{I}\vee(\mathcal{R}'\subseteq\mathcal{R})$, $\mathcal{I}'\rightarrow\mathcal{I}$

$\underline{\mathcal{I}\vee(\mathcal{M}\subseteq\mathcal{R})}$**:** `TheoryUpdate`$_i$ does nothing, so $\mathcal{M}=\emptyset$.

A.5.13 Theory-Specific Code for Shostak Theory $\mathcal{T}_\chi$

TheoryAddSharedTerm and TheoryAssert

These are exactly the same as for the Nelson-Oppen theory-specific code, shown above.

TheoryCheckSat

Line 320:

```
...   [ X := ∅; ]
318.  G, P=(all), P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find]
319.  FOREACH e in Aχ DO BEGIN
320.    G, P=(all), P1[Φ,H,I,N], e ∈ Aχ, S2[I,S,*.find], S3[A,I,*.find],
        S29[I,S](X,χ)
...
326.    G, P=(all), P1[Φ,H,I,N], S2[I,S,*.find], S3[A,I,*.find],
        S29[I,S](X,χ)
327.  END
```

$\underline{S_{29}[\mathcal{I},\mathcal{S}](X,\chi)}$**:** From 318: $X = \emptyset$. From 326: by $S_{29}[\mathcal{I},\mathcal{S}](X,\chi)$ at 326.

Line 322:

```
320.    G, P=(all), P1[Φ,H,I,N], e ∈ Aχ, S2[I,S,*.find], S3[A,I,*.find],
        S29[I,S](X,χ)
321.    IF Op(e) = '¬' AND Find(e[1][1]) ≡ Find(e[1][2]) THEN BEGIN
322.      G, P=(all), P1[Φ,H,I,N], Op(e[1]) = '=', Op(e) = '¬',
          e[1][1] ~ e[1][2], e ∈ Aχ, S2[I,S,*.find]
...
237. G, hf(t)
238. Find(t)
...
250. END Find
251. G, P=(all), retval.find ≡ retval, t ~ retval
```

We first consider the preconditions of the calls to `Find`. We assume that the calls to `Find` are not made unless $Op(e) =$ '¬' (i.e. C-like semantics). Since $e \in \mathcal{A}$, we know e is a literal by G_{20}. Now, since e is in $\mathcal{A}_\chi$, e is a χ-literal, so $e[1]$ must be an equation between two terms since Shostak theories do not have predicate symbols. Finally, we know $hf(e[1][1])$ and $hf(e[1][2])$ by G_{11} at 320.

$\underline{G}$**:** By G at 320 and 251.

$\underline{P_=(all)}$**:** By $P_=(all)$ at 320 and 251.

$\underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]}$**:** By $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ at 320 and $P_=(all)$.

$\underline{Op(e[1]) = \text{`='}}$: See above paragraph.

$\underline{Op(e) = \text{'}\neg\text{'}}$: By the if-condition.

$\underline{e[1][1] \sim e[1][2]}$: By $t \sim \mathit{retval}$ at 251, the if-condition, and the definition of $\sim$.

$\underline{e \in \mathcal{A}_\chi}$: By $e \in \mathcal{A}_\chi$ at 320 and $P_=(\mathit{all})$.

$\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$: By $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$ at 320 and $P_=(\mathit{all})$.

Line 324:

```
322.    G, P_=(all), P_1[Φ,H,I,N], Op(e[1]) = '=', Op(e) = '¬',
        e[1][1] ~ e[1][2], e ∈ A_χ, S_2[I,S,*.find]
323.    I := TRUE; RETURN;
324.    G, P_=(all − {I,N,Q}), P_1[Φ,H,I,N], P_9[Φ,I,N,Q,Λ_*,*.find](χ),
        S_2[I,S,*.find], S_3[A,I,*.find]
```

$\underline{G}$: Only G_1 depends on $\mathcal{I}$. Since line 284 sets $\mathcal{I}$ to be `TRUE`, we must show that $\mathcal{T} \cup \mathcal{H} \models \mathit{false}$. Let $s \equiv e[1][1]$ and $t \equiv e[1][2]$. We know that $s \sim t$, so by the definition of $\sim$ and $S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]$ at 322, $\mathit{canon}_\chi(\mathcal{S}(s)) \equiv \mathit{canon}_\chi(\mathcal{S}(t))$. Then, by property 1 of *canon*, $\mathcal{T}_i \models \gamma_\chi(\mathcal{S}(s) = \mathcal{S}(t))$. Then, by Proposition 2.1, $\mathcal{T}_i \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(s = t)$, and thus $\mathcal{T} \cup \mathcal{S} \models s = t$. But $\mathcal{T} \cup \mathcal{F} \models \mathcal{S}$ by G_{14} and $\mathcal{T} \cup \Phi \models \mathcal{F}$ by G_2, so $\mathcal{T} \cup \Phi \models s = t$. But $s \neq t \in \mathcal{A} \subseteq \Phi$, so $\mathcal{T} \cup \Phi \models \mathit{false}$. Thus, by G_3, $\mathcal{T} \cup \mathcal{H} \models \mathit{false}$.

$\underline{P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]}$: Trivial since $\mathcal{I} =$ `TRUE`.

$\underline{P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.\mathit{find}](\chi)}$: Trivial since $\mathcal{I} =$ `TRUE`.

$\underline{S_2[\mathcal{I}, \mathcal{S}, *.\mathit{find}]}$: Trivial since $\mathcal{I} =$ `TRUE`.

$\underline{S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}]}$: Trivial since $\mathcal{I} =$ `TRUE`.

Line 326:

320. G, $P_{=}(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $e \in \mathcal{A}_\chi$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$, $S_{29}[\mathcal{I}, \mathcal{S}](X, \chi)$
321. IF $Op(e) =$ '¬' AND Find(e[1][1]) $\equiv$ Find(e[1][2]) THEN BEGIN
...
RETURN;
...
325. END [$X := X \cup \{e\}$;]
326. G, $P_{=}(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$, $S_{29}[\mathcal{I}, \mathcal{S}](X, \chi)$

$S_{29}[\mathcal{I}, \mathcal{S}](X, \chi)$: Assume $\neg\mathcal{I}$, $d \in X$, and $Op(d) \neq$ '¬'. We must show $\mathcal{T}_i \cup \gamma_i(\mathcal{S}) \not\models \gamma_i(d[1])$. If $d \in X_{320}$, then this follows by $S_{29}[\mathcal{I}, \mathcal{S}](X, \chi)$ at 320. Otherwise, $d \equiv e$, and thus $d[1]$ is an equation and $d[1][1] \not\sim d[1][2]$ (see comments for line 322, above). Let $s \equiv d[1][1]$ and $t \equiv d[1][2]$. Since $s \not\sim t$, it follows from $S_2[\mathcal{I}, \mathcal{S}, *.find]$ that $canon_\chi(\mathcal{S}(s)) \not\equiv canon_\chi(\mathcal{S}(t))$. Thus, by property 1 of *canon*, $\mathcal{T}_i \not\models \gamma_\chi(\mathcal{S}(s) = \mathcal{S}(t))$, and thus, by Proposition 2.1, $\mathcal{T}_i \cup \gamma_\chi(\mathcal{S}) \not\models \gamma_\chi(s = t)$.

Line 328:

... [$X := \emptyset$;]
318. G, $P_{=}(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$
319. FOREACH e in $\mathcal{A}_\chi$ DO BEGIN
...
326. G, $P_{=}(all)$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$, $S_{29}[\mathcal{I}, \mathcal{S}](X, \chi)$
327. END
328. G, $P_{=}(all - \{\mathcal{I}, \mathcal{N}, \mathcal{Q}\})$, $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](\chi)$, $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $S_3[\mathcal{A}, \mathcal{I}, *.find]$

$P_9[\Phi, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \Lambda_*, *.find](\chi)$: Suppose $\mathcal{Q} = \emptyset$, $\neg\mathcal{I}$, and *convex*. We must show that $\mathcal{T}_\chi \cup \gamma_\chi(\Phi_\chi \cup Ar_{\sim_\chi})$ is satisfiable. First, note that since $\mathcal{A}_\chi$ contains only χ-literals (by definition and by G_{20}), every formula in $\mathcal{A}_\chi$ is either an equation or a disequation. Let $\mathcal{A}_\chi^{=}$ be the set of all equations in $\mathcal{A}$ and $\mathcal{A}_\chi^{\neq}$ the set of all disequations in $\mathcal{A}_\chi$. Now, $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S})$ is satisfiable by Corollary 2.1 since $\mathcal{S}$ is in χ-solved form by G_{13}. We next show that $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S} \cup \mathcal{A}_\chi^{\neq} \cup D_{\sim_\chi})$ is satisfiable.

Suppose it is not, then $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(\neg(\mathcal{A}^{\neq}_\chi \cup D_{\sim_\chi}))$. But then, since $\mathcal{T}_\chi$ is convex, it must be the case that $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(s = t)$ where $s \neq t \in (\mathcal{A}^{\neq}_\chi \cup D_{\sim_\chi})$. But if $s \neq t \in \mathcal{A}^{\neq}_\chi$, then by $S_{29}[\mathcal{I}, \mathcal{S}](X, \chi)$ (which holds vacuously at 318 where $X = \emptyset$) and the end-of-loop condition, $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \not\models \gamma_i(s = t)$. Suppose on the other hand that $s \neq t \in D_{\sim_\chi}$. By definition of $D_{\sim_\chi}$, $hf(s) \wedge hf(t) \wedge find^*(s) \not\equiv find^*(t)$, so by $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $canon_\chi(\mathcal{S}(s)) \not\equiv canon_\chi(\mathcal{S}(t))$. Then, by property 1 of *canon* and Proposition 2.1, $\mathcal{T}_i \cup \gamma_\chi(\mathcal{S}) \not\models \gamma_\chi(s = t)$. Thus, $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S} \cup \mathcal{A}^{\neq}_\chi \cup D_{\sim_\chi})$ is satisfiable. Finally, suppose $s = t \in \mathcal{A}^{=}_\chi \cup E_{\sim_\chi}$. Then, by $S_3[\mathcal{A}, \mathcal{I}, *.find]$ or by the definition of $E_{\sim_\chi}$, $hf(s) \wedge hf(t) \wedge find^*(s) \equiv find^*(t)$, and thus, by $S_2[\mathcal{I}, \mathcal{S}, *.find]$, $canon_\chi(\mathcal{S}(s)) \equiv canon_\chi(\mathcal{S}(t))$. It then follows by property 1 of *canon* and Proposition 2.1 that $\mathcal{T}_i \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(s = t)$. Thus, $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S} \cup \mathcal{A}_\chi \cup Ar_{\sim_\chi})$ is satisfiable. But $\mathcal{B}_\chi = \emptyset$ by G_{12}, so it follows that $\mathcal{T}_\chi \cup \gamma_\chi(\Phi_\chi \cup Ar_{\sim_\chi})$ is satisfiable.

TheoryRewrite

Line 335:

```
333.   G, P_=(all), free(e) ⊆ V, fr(e), P_33(e), S_25[S, *.find](e)
334.   IF e is not a term THEN BEGIN
335.     G, P_=(all), free(e) ⊆ V, fr(e), P_33(e), ¬P_23(e), S_4[S, *.find](e)
```

$\underline{\neg P_{23}(e)}$**:** By the if-condition.

$\underline{S_4[\mathcal{S}, *.find](e)}$**:** Since e is not a term, if a term $t \trianglelefteq e$, then $t \trianglelefteq c$ for some child c of e. $S_4[\mathcal{S}, *.find](e)$ then follows from $S_{25}[\mathcal{S}, *.find](e)$.

Line 337:

```
335.     G, P_=(all), free(e) ⊆ V, fr(e), P_33(e), ¬P_23(e), S_4[S, *.find](e)
336.     RETURN e;
337.     G, T ∪ Φ |= e ≃ retval, fr(retval), free(retval) ⊆ V, P_34[all](χ),
         S_26[S, *.find](e, retval), S_42[S](e, retval)
```

$\underline{P_{34}[all](\chi)}$**:** By $P_=(all)$ and Lemma A.5.

$\underline{S_{26}[\mathcal{S}, *.find](e, retval)}$**:** By $S_4[\mathcal{S}, *.find](e)$ since $e \equiv retval$.

$\underline{S_{42}[\mathcal{S}](e, \mathit{retval})}$: e is not a term by $\neg P_{23}(e)$ at 335.

Line 339:

333. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathit{fr}(e),\ P_{33}(e),\ S_{25}[\mathcal{S}, *.\mathit{find}](e)$
334. `IF` e `is not a term THEN BEGIN`
...
336. `RETURN` e`;`
...
338. `END`
339. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathit{fr}(e),\ \underline{P_{23}(e)},\ S_{25}[\mathcal{S}, *.\mathit{find}](e)$

$\underline{P_{23}(e)}$: By the if-condition.

Line 341:

339. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(e) \subseteq \mathcal{V},\ \mathit{fr}(e),\ P_{23}(e),\ S_{25}[\mathcal{S}, *.\mathit{find}](e)$
340. e^* `:= RewriteHelper(`e`);`
341. $\underline{G,\ P_{=}(\mathit{all}),\ \mathit{free}(e^*) \subseteq \mathcal{V},\ P_{23}(e^*),\ \mathcal{T} \cup \Phi \models e \simeq e^*,\ S_{30}[*.\mathit{find}](e^*),\ S_{31}[\mathcal{S}, *.\mathit{find}](e^*),\ S_{32}[\mathcal{S}](e^*),\ S_{33}[\mathcal{S}](e^*, e)}$
...
381. $G,\ \mathit{free}(t) \subseteq \mathcal{V},\ P_{23}(t),\ \mathit{fr}(t) \vee \mathit{hf}(t),\ S_{25}[\mathcal{S}, *.\mathit{find}](t)$
382. `RewriteHelper(`t`)`
...
410. `END RewriteHelper`
411. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(\mathit{retval}) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models t' \simeq \mathit{retval},\ P_{23}(\mathit{retval}),\ S_{30}[*.\mathit{find}](\mathit{retval}),\ S_{31}[\mathcal{S}, *.\mathit{find}](\mathit{retval}),\ S_{32}[\mathcal{S}](\mathit{retval}),\ S_{33}[\mathcal{S}](\mathit{retval}, t')$

$\underline{G}$: By G at 411.

$\underline{P_{=}(\mathit{all})}$: By $P_{=}(\mathit{all})$ at 339 and 411.

$\underline{\mathit{free}(e^*) \subseteq \mathcal{V}}$: By $\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}$ at 411.

$\underline{P_{23}(e^*)}$: By $P_{23}(\mathit{retval})$ at 411.

$\underline{\mathcal{T} \cup \Phi \models e \simeq e^*}$: By $\mathcal{T} \cup \Phi \models t' \simeq \mathit{retval}$ at 411.

$\underline{S_{30}[*.\mathit{find}](e^*)}$: By $S_{30}[*.\mathit{find}](\mathit{retval})$ at 411.

$\underline{S_{31}[\mathcal{S}, *.\mathit{find}](e^*)}$: By $S_{31}[\mathcal{S}, *.\mathit{find}](\mathit{retval})$ at 411.

$\underline{S_{32}[\mathcal{S}](e^*)}$**:** By $S_{32}[\mathcal{S}](\mathit{retval})$ at 411.

$\underline{S_{33}[\mathcal{S}](e^*, e)}$**:** By $S_{33}[\mathcal{S}](\mathit{retval}, t')$ at 411.

Line 343:

341. G, $P_{=}(\mathit{all})$, $\mathit{free}(e^*) \subseteq \mathcal{V}$, $P_{23}(e^*)$, $\mathcal{T} \cup \Phi \models e \simeq e^*$, $S_{30}[*.\mathit{find}](e^*)$, $S_{31}[\mathcal{S}, *.\mathit{find}](e^*)$, $S_{32}[\mathcal{S}](e^*)$, $S_{33}[\mathcal{S}](e^*, e)$

342. `RETURN` canon_{χ}`(`e^*`);`

343. G, $\underline{\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}}$, $\underline{\mathit{fr}(\mathit{retval})}$, $\underline{\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}}$, $\underline{P_{34}[\mathit{all}](\chi)}$, $\underline{S_{26}[\mathcal{S}, *.\mathit{find}](e, \mathit{retval})}$, $\underline{S_{42}[\mathcal{S}](e, \mathit{retval})}$

Note that the only requirement needed to be able to apply canon_{χ} to an expression is that the expression be a term. This is guaranteed by $P_{23}(e^*)$ at 341.

$\underline{\mathcal{T} \cup \Phi \models e \simeq \mathit{retval}}$**:** By $\mathcal{T} \cup \Phi \models e \simeq e^*$ at 341 and properties 1 and 2 of *canon*.

$\underline{\mathit{fr}(\mathit{retval})}$**:** By $S_{30}[*.\mathit{find}](e^*)$ at 341 and property 3 of *canon*, we know that $\forall c \in \delta_{\chi}(\mathit{retval}).\mathit{fr}(c)$. Then, by G_{18}, it follows that $\mathit{fr}(\mathit{retval})$.

$\underline{\mathit{free}(\mathit{retval}) \subseteq \mathcal{V}}$**:** By $\mathit{free}(e^*) \subseteq \mathcal{V}$ at 341 and property 3 of *canon*.

$\underline{P_{34}[\mathit{all}](\chi)}$**:** By $P_{=}(\mathit{all})$ at 341 and Lemma A.5.

$\underline{S_{26}[\mathcal{S}, *.\mathit{find}](e, \mathit{retval})}$**:** We will show $S_4[\mathcal{S}, *.\mathit{find}](\mathit{retval})$. It is easy to see that this property implies $S_{26}[\mathcal{S}, *.\mathit{find}](e, \mathit{retval})$. Consider $t \trianglelefteq \mathit{retval}$, and suppose $\neg \mathit{hf}(t)$. We must show $t \equiv \mathit{canon}_{\chi}(\mathcal{S}(t))$. First suppose that $t \trianglelefteq c$ for some $c \in \delta_{\chi}(\mathit{retval})$. We know that $c \in \delta_{\chi}(e^*)$ by property 3 of *canon*. Then, by $S_{31}[\mathcal{S}, *.\mathit{find}](e^*)$ at 341, we have $S_4[\mathcal{S}, *.\mathit{find}](c)$. Thus, since $t \trianglelefteq c$, $t \equiv \mathit{canon}_{\chi}(\mathcal{S}(t))$. Suppose on the other hand that $t \not\trianglelefteq c$ for any $c \in \delta_{\chi}(\mathit{retval})$. Then t must be a compound χ-term an it must be the case that $\delta_{\chi}(t) \subseteq \delta_{\chi}(\mathit{retval})$. But again by property 3 of *canon*, $\delta_{\chi}(\mathit{retval}) \subseteq \delta_{\chi}(\mathit{retval})$, so $\delta_{\chi}(t) \subseteq \delta_{\chi}(e^*)$. Thus, by $S_{32}[\mathcal{S}](e^*)$ at 341, $\mathcal{S}(t) \equiv t$. Also, by property 5 of *canon*, $\mathit{canon}_{\chi}(t) \equiv t$, and thus $t \equiv \mathit{canon}_{\chi}(\mathcal{S}(t))$.

$\underline{S_{42}[\mathcal{S}](e, \mathit{retval})}$**:** By $S_{33}[\mathcal{S}](e^*, e)$ at 341, $\mathit{canon}_{\chi}(e^*) \equiv \mathit{canon}_{\chi}(\mathcal{S}(e))$.

TheorySolve

Line 350:

348. G, $P_{=}(all)$, $P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](e)$, $\neg\mathcal{I}$, $P_6[\Phi,\mathcal{H}](e)$, $free(e) \subseteq \mathcal{V}$, $fr(e)$, $P_{11}(e)$, $Op(e) = \text{`='}$, $e[1] \not\equiv e[2]$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $S_3[\mathcal{A},\mathcal{I},*.find]$, $S_4[\mathcal{S},*.find](e)$

349. `RETURN` $solve_\chi(e)$; [$\mathcal{V}$:= $\mathcal{V}$ $\cup$ $free(retval)$]

350. G, $P_{=}(\Phi,\mathcal{H},\mathcal{I},\mathcal{N},*.find)$, $free(retval) \subseteq \mathcal{V}$, $\underline{P_{14}[*.find](retval)}$, $\underline{P_{17}[\Phi](\Phi')}$, $\underline{P_{35}[\Phi](e,retval)}$, $S_2[\mathcal{I},\mathcal{S},*.find]$, $\underline{S_3[\mathcal{A},\mathcal{I},*.find]}$, $\underline{S_7[\mathcal{S},*.find](retval)}$

Note that by the definition of *solve*, either $retval = \{false\}$, $retval = \emptyset$, or $retval$ is a set of equations in χ-solved form. In each case, we have $\mathcal{T}_\chi \models \gamma_\chi(e \leftrightarrow \exists\overline{w}.retval)$, where $\overline{w} = free(retval) - free(e)$ and $\overline{w} \cap \mathcal{V}' = \emptyset$ (recall that when used as a formula, the empty set is equivalent to $true$). Also, if $retval$ is a set of equations, then $canon_\chi(d[2]) \equiv d[2]$ for each $d \in retval$.

$\underline{P_{14}[*.find](retval)}$**:** Suppose $retval \neq \{false\}$. If $retval = \emptyset$, then $\forall\, e \in retval.\, fr(e)$ and $P_{13}(retval)$ follow trivially. Otherwise, $P_{13}(retval)$ follows from the fact that $retval$ is in χ-solved form. It remains to show $\forall\, e \in retval.\, fr(e)$. We first show $\forall\, c \in \delta_\chi(retval).\, fr(c)$. First note that by the definition of *solve*, $\delta_\chi(retval) \subseteq \delta_\chi(e) \cup \overline{w}$. Also, the variables in $\overline{w}$ are fresh, so $\overline{w} \cap HF = \emptyset$. Suppose $c \in \delta_\chi(d)$. If $c \in \overline{w}$, then c has no children and $\neg hf(c)$, so it follows that $fr(c)$. If $c \notin \overline{w}$, then $c \in \delta_\chi(e)$. We consider two cases.

1. Suppose $\neg hf(c)$. c is a sub-term of e, so by G_6 there is a path from e to c such that for every expression d in the path, $\neg hf(d)$. Thus, every highest find-initialized sub-expression of d is also a highest find-initialized sub-expression of e. Thus, since e is find-reduced, so is c.

2. Suppose $hf(c)$, and suppose c is not find-reduced, so that $c.find \not\equiv c$. Then because $c \in \delta_\chi(e)$ and e is find-reduced, there must a term $t \trianglelefteq e$ such that $c \in \delta_\chi(t)$ and $t.find \equiv t$. Then, by $\neg\mathcal{I}$ and $S_2[\mathcal{I},\mathcal{S},*.find]$, $t \equiv canon_\chi(\mathcal{S}(t))$. Now, since c is a χ-leaf and $hf(c)$ and $c.find \not\equiv c$, it follows from G_{19} that $c \in lhs(\mathcal{S})$, and thus $c \notin \delta_\chi(\mathcal{S}(t))$. But $\delta_\chi(canon_\chi(\mathcal{S}(t))) \subseteq$

$\delta_\chi(\mathcal{S}(t))$ by property 3 of *canon*, so $c \notin \delta_\chi(canon_\chi(\mathcal{S}(t)))$ which is a contradiction since $canon_\chi(\mathcal{S}(t)) \equiv t$.

We have shown that $\forall\, c \in \delta_\chi(retval).\, fr(c)$. Now, suppose $d \in retval$. We know that $\forall\, c \in \delta_\chi(d).\, fr(c)$. Thus, by G_{18}, $fr(d)$.

$P_{17}[\Phi](\Phi')$**:** Φ is unchanged by 349, so $\Phi = \Phi'$.

$P_{35}[\Phi](e, retval)$**:** We have $\mathcal{T}_\chi \models \gamma_\chi(e \leftrightarrow \exists \overline{w}.retval)$, where $\overline{w} = free(retval) - free(e)$ and $\overline{w} \cap \mathcal{V}' = \emptyset$. It follows that $\mathcal{T} \cup \Phi \models e \leftrightarrow \exists \overline{w}.retval$. Also, $free(\Phi) \subseteq \mathcal{V}'$ by $P_=(\Phi)$ at 350, and $P_=(all)$ and G_8 at 348, so $\overline{w} \cap (\mathcal{V}' \cup free(\Phi)) = \emptyset$.

$S_7[\mathcal{S}, *.find](retval)$**:** Suppose $retval \neq \{false\}$. If $retval = \emptyset$, then $S_4[\mathcal{S}, *.find](retval)$ and $S_6(retval)$ follow trivially. Otherwise, $S_6(retval)$ clearly holds since $retval$ is in χ-solved form. It remains to show $S_4[\mathcal{S}, *.find](retval)$. Consider $t \trianglelefteq retval$. There are two cases.

1. Suppose $t \trianglelefteq c$ for some $c \in \delta_\chi(retval)$. Then, as shown above, $\delta_\chi(retval) \subseteq \delta_\chi(e) \cup \overline{w}$, so $t \in \overline{w}$ or $t \triangleleft c$ for some $c \in \delta_\chi(e)$. If $t \in \overline{w}$, then t is a χ-leaf and $\neg hf(t)$, and thus $t \notdownarrow \mathcal{S}$ by G_{19}, so $\mathcal{S}(t) \equiv t$. Also, by property 4 of *canon*, $canon_\chi(t) \equiv t$, so $canon_\chi(\mathcal{S}(t)) \equiv t$. If $t \triangleleft c$ for some $c \in \delta_\chi(e)$, then $canon_\chi(\mathcal{S}(t)) \equiv t$ by $S_4[\mathcal{S}, *.find](e)$ at 348.

2. Suppose $t \not\trianglelefteq c$ for any $c \in \delta_\chi(retval)$ and suppose $t \trianglelefteq d$ where $d \in retval$. Then $t \not\trianglelefteq d[1]$ since $retval$ is in χ-solved form and thus $d[1]$ is a χ-leaf. Thus t must be a compound χ-term and there must be a path of compound χ-terms from $d[2]$ to t. Clearly, then, $\delta_\chi(t) \subseteq \delta_\chi(d[2])$. Now, as shown above, $\forall\, c \in \delta_\chi(retval).\, fr(c)$. It follows that $\forall\, c \in \delta_\chi(retval).\, \mathcal{S}(c) \equiv c$. To see this, notice that if $\neg hf(c)$, then $c \notdownarrow \mathcal{S}$ by G_{19}, and if $hf(c)$, then since $c.find \equiv c$, it follows (also by G_{19}) that $c \notin lhs(\mathcal{S})$. Thus, $\mathcal{S}(t) \equiv t$. Also, since there is a path of compound χ-terms from $d[2]$ to t and we know that $canon_\chi(d[2]) \equiv d[2]$, it follows by repeated application of property 5 of *canon* that $canon_\chi(t) \equiv t$. Thus $canon_\chi(\mathcal{S}(t)) \equiv t$.

TheorySetup

Line 357:

```
355.   G, P_=(all), hf(e)
356.   IF e is a compound χ-term THEN BEGIN [ Z := ∅; ]
357.     G, P_=(all), hf(e), S_8(e)
```

$\underline{S_8(e)}$**:** By the if-condition.

Line 359:

```
...      [ Z := ∅; ]
357.     G, P_=(all), hf(e), S_8(e)
358.     FOREACH c ∈ δ_χ(e) DO BEGIN
359.       G, P_=(all − {*.notify}), P_⊆(*.notify), hf(e), S_8(e), S_36[*.notify](Z)
...
361.       G, P_=(all − {*.notify}), P_⊆(*.notify), hf(e), S_8(e), S_36[*.notify](Z)
362.     END
```

$\underline{S_{36}[*.notify](Z)}$**:** From 357: trivially true since $Z = \emptyset$. From 361: by $S_{36}[*.notify](Z)$ at 361.

Line 361:

```
359.       G, P_=(all − {*.notify}), P_⊆(*.notify), hf(e), S_8(e), S_36[*.notify](Z)
360.       c.notify := c.notify ∪ {(χ, e)}; [ Z := Z ∪ {c}; ]
361.       G, P_=(all − {*.notify}), P_⊆(*.notify), hf(e), S_8(e), S_36[*.notify](Z)
```

$\underline{G}$**:** We must consider only those global properties which depend on $*.notify$.

$\underline{G_{16}}$**:** By G_{16} at 359 and $*.notify_{359} \subseteq *.notify$.

$\underline{G_{17}}$**:** By G_{17}, $S_8(e)$, $hf(e)$ and $c \in \delta_\chi(e)$ at 359.

$\underline{S_{36}[*.notify](Z)}$**:** By $S_{36}[*.notify](Z)$ at 359, the execution of line 360, and $Z = Z_{359} \cup \{c\}$.

Line 363:

```
357.     G, P_=(all), hf(e), S_8(e)
358.     FOREACH c ∈ δ_χ(e) DO BEGIN
...
361.       G, P_=(all − {∗.notify}), P_⊆(∗.notify), hf(e), S_8(e), S_36[∗.notify](Z)
362.     END
363.     G, P_34[all](χ), S_35[∗.notify](e)
```

$\underline{P_{34}[all](\chi)}$**:** From 357: by $P_{=}(all)$ at 357 and Lemma A.5. From 361: by $P_{=}(all - \{*.notify\})$ and $P_{\subseteq}(*.notify)$ at 361 and a similar argument as that found in Lemma A.5.

$\underline{S_{35}[*.notify](e)}$**:** From 358: holds vacuously since $\delta_{\chi}(e) = \emptyset$. From 361: by $S_8(e)$ and $S_{36}[*.notify](Z)$ at 361 and the end-of-loop condition, $Z = \delta_{\chi}(e)$.

Line 365:

```
355.   G, P_=(all), hf(e)
356.   IF e is a compound χ-term THEN BEGIN
...
363.     G, P_34[all](χ), S_35[∗.notify](e)
364.   END
365.   G, P_34[all](χ), S_35[∗.notify](e)
```

$\underline{P_{34}[all](\chi)}$**:** From 355: by $P_{=}(all)$ and Lemma A.5. From 363: trivial.

$\underline{S_{35}[*.notify](e)}$**:** From 355: $\neg S_8(e)$ by the if-condition. From 363: trivial.

TheoryUpdate

Line 374:

372. $G,\ P_{=}(all),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \neg\mathcal{I},\ d.find \equiv d,\ S_3[\mathcal{A},\mathcal{I},*.find],\ S_8(d),\ S_9[*.find](d)$

373. d^* := TheoryRewrite$_\chi(d)$;

374. $G,\ P_{=}(\mathcal{S},*.find),\ P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}],\ \neg\mathcal{I},\ \neg\mathcal{I}',\ fr(d^*),\ P_{23}(d^*),\ d.find \equiv d,\ S_1[\mathcal{S}](d^*,d),\ S_3[\mathcal{A},\mathcal{I},*.find],\ S_4[\mathcal{S},*.find](d^*),\ S_8(d),\ S_9[*.find](d)$

...

331. $G,\ free(e) \subseteq \mathcal{V},\ fr(e),\ P_{33}(e),\ S_{25}[\mathcal{S},*.find](e)$

332. TheoryRewrite$_\chi(e)$

...

344. END TheoryRewrite$_\chi$

345. $G,\ \mathcal{T} \cup \Phi \models e \simeq retval,\ fr(retval),\ free(retval) \subseteq \mathcal{V},\ P_{34}[all](\chi),\ S_{26}[\mathcal{S},*.find](e,retval),\ S_{42}[\mathcal{S}](e,retval)$

We first consider the preconditions of TheoryRewrite$_\chi$.

$free(d) \subseteq \mathcal{V}$**:** By $d.find \equiv d$ at 372, $d \lhd \mathcal{F}$, so by G_8, $free(d) \subseteq \mathcal{V}$.

$fr(d)$**:** By $d.find \equiv d$ at 372, $d.find \equiv d$, so $fr(d)$.

$P_{33}(d)$**:** By $d.find \equiv d$ at 372, $hf(d)$, so by G_7, d is a term.

$S_{25}[\mathcal{S},*.find](d)$**:** By $d.find \equiv d$ at 372, $hf(d)$, so by G_6, if $t \trianglelefteq d$, $hf(t)$. It follows that $S_4[\mathcal{S},*.find](d)$ and thus also $S_{25}[\mathcal{S},*.find](d)$ at 372.

We now consider the properties at 374. Recall that $P_{34}[all](i)$ at 345 implies $P_{=}(\Phi - \mathcal{B}_i,\mathcal{H},\mathcal{I},\mathcal{N},\mathcal{S},\Lambda_*,*.find)$, $P_{\subseteq}(\mathcal{Q},\mathcal{B}_i,\mathcal{V},*.notify)$, $\mathcal{T}_i \cup \gamma_i(\Phi_i' \models \exists\,\overline{w}.\,\mathcal{B}_i) \wedge \overline{w} \cap \mathcal{V}' = \emptyset \wedge \overline{w} \subseteq \mathcal{V}$, where $\overline{w} = free(\mathcal{B}_i) - free(\Phi_i')$, and $\mathcal{T}_i \cup \gamma_i(\Phi_i \models (\mathcal{Q} - \mathcal{Q}')) \wedge free(\mathcal{Q} - \mathcal{Q}') \subseteq \mathcal{V}$.

G**:** By G at 345.

$P_{=}(\mathcal{R},\mathcal{S},*.find)$**:** By $P_{=}(all)$ at 372 and $P_{=}(Eq,*.find)$ at 345.

$P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$**:** By $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ at 372 and $P_{=}(\mathcal{H},\mathcal{I},\mathcal{N})$ and $P_{\subseteq}(\Phi)$ at 345.

$\neg\mathcal{I}$**:** By $\neg\mathcal{I}$ at 372 and $P_{=}(\mathcal{I})$ at 345.

$\neg\mathcal{I}'$**:** By $\neg\mathcal{I}$ and $P_{=}(all)$ at 372.

$\underline{fr(d^*)}$: By $fr(retval)$ at 345.

$\underline{d.find \equiv d}$: By $d.find \equiv d$ at 372 and $P_{=}(*.find)$ at 345.

$\underline{S_1[\mathcal{S}](d^*, d)}$: By $d.find \equiv d$ and G_7 at 372, d is a term, so by $S_{42}[\mathcal{S}](e, retval)$ at 345, $d^* \equiv canon_\chi(\mathcal{S}(d))$.

$\underline{P_{23}(d^*)}$: Since $d^* \equiv canon_\chi(\mathcal{S}(d))$, and $canon$ is a function from terms to terms, d^* is a term.

$\underline{S_3[\mathcal{A}, \mathcal{I}, *.find]}$: By $S_3[\mathcal{A}, \mathcal{I}, *.find]$ at 372, $P_{=}(\mathcal{A}, \mathcal{I}, *.find)$ at 345.

$\underline{S_4[\mathcal{S}, *.find](d^*)}$: Suppose $t \trianglelefteq d^*$ and $\neg hf(t)$. If $t \not\equiv d^*$, then $t \equiv canon_\chi(\mathcal{S}(t))$ by $S_{26}[\mathcal{S}, *.find](e, retval)$ at 345. Otherwise, we know that $t \equiv canon_\chi(\mathcal{S}(d))$ (shown above). So, by Lemma A.1, $\mathcal{S}(t) \equiv t$. Furthermore, by property 2 of $canon$, $canon_\chi(t) \equiv t$. Thus, $t \equiv canon_\chi(\mathcal{S}(t))$.

$\underline{S_8(d)}$: By $S_8(d)$ at 372.

$\underline{S_9[*.find](d)}$: By $S_9[*.find](d)$ at 372 and $P_{=}(*.find)$ at 345.

Line 376:

374. $G,\ P_{=}(\mathcal{S}, *.find),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \neg\mathcal{I},\ \neg\mathcal{I}',\ fr(d^*),\ P_{23}(d^*),\ d.find \equiv d,$ $S_1[\mathcal{S}](d^*, d),\ S_3[\mathcal{A}, \mathcal{I}, *.find],\ S_4[\mathcal{S}, *.find](d^*),\ S_8(d),\ S_9[*.find](d)$

375. `AssertEqualities(`$\{d = d^*\}$`);`

376. $\underline{G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \neg\mathcal{I}',\ S_3[\mathcal{A}, \mathcal{I}, *.find],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),}$ $\underline{\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R}),\ \mathcal{I} \vee (d \in \mathcal{R})}$

...

58. $G,\ \underline{P_4[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}](\mathcal{E})},\ \neg\mathcal{I},\ \underline{P_{12}[\Phi, \mathcal{H}](\mathcal{E})},\ \underline{P_{14}[*.find](\mathcal{E})},\ \underline{free(\mathcal{E}) \subseteq \mathcal{V}},$ $S_3[\mathcal{A}, \mathcal{I}, *.find],\ \underline{S_{12}[\mathcal{S}, *.find](\mathcal{E})}$

59. `AssertEqualities(`$\mathcal{E}$`)`

...

97. `END AssertEqualities`

98. $G,\ P_{\subseteq}(\mathcal{F}, lhs(\mathcal{S})),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ S_3[\mathcal{A}, \mathcal{I}, *.find],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),$ $\mathcal{I} \vee (lhs(\mathcal{E}) \subseteq \mathcal{R}),\ \mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$

We first consider the preconditions of `AssertEqualities`. First note that since $d^* \equiv canon_\chi(\mathcal{S}(d))$, $\mathcal{T} \cup \mathcal{S} \models d = d^*$. Then, since $\mathcal{T} \cup \Phi \models \mathcal{F}$ by G_2 and $\mathcal{T} \cup \mathcal{F} \models \mathcal{S}$ by G_{14}, it follows that $\mathcal{T} \cup \Phi \models d = d^*$.

$\underline{P_4[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}](\{d=d^*\})}$**:** By $\neg\mathcal{I}$ and $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$, $\mathcal{T}\cup\mathcal{N}\cup\Phi\models\mathcal{H}$. Then, since $\mathcal{T}\cup\Phi\models d=d^*$, it follows that $\neg\mathcal{I}\rightarrow(\mathcal{T}\cup\mathcal{N}\cup\Phi\cup\{d=d^*\}\models\mathcal{H})$.

$\underline{P_{12}[\Phi,\mathcal{H}](\{d=d^*\})}$**:** Trivial, since as mentioned above, $\mathcal{T}\cup\Phi\models d=d^*$.

$\underline{P_{14}[*.\mathit{find}](\{d=d^*\})}$**:** $P_{13}(\{d=d^*\})$ is true by construction (note that d^* is a term by $P_{23}(d^*)$ and d is a term by $d.\mathit{find}\equiv d$ and G_7). Then, $\forall\, e\in\{d=d^*\}.\, \mathit{fr}(e)$ follows by $d.\mathit{find}\equiv d$ and $\mathit{fr}(d^*)$.

$\underline{\mathit{free}(\{d=d^*\})\subseteq\mathcal{V}}$**:** Suppose c is a χ-leaf in $d=d^*$. We will show $\mathit{hf}(c)$. Then, since every free variable is also a χ-leaf, $\mathit{free}(\{d=d^*\})\subseteq\mathcal{V}$ follows by G_8. To show $\mathit{hf}(c)$, we consider two cases. Suppose first that $c\trianglelefteq d$. Then since $d.\mathit{find}\equiv d$, it follows from G_6 that $\mathit{hf}(c)$. Suppose on the other hand that $c\trianglelefteq d^*$. We know that $d^*\equiv \mathit{canon}_\chi(\mathcal{S}(d))$. By property 3 of *canon*, $c\in\delta_\chi(\mathcal{S}(d))$. It follows that $c\in\delta_\chi(d)$ or $c\in\delta_\chi(\mathcal{S})$. But if $c\in\delta_\chi(d)$, then as we just showed, $\mathit{hf}(c)$. If $c\in\delta_\chi(\mathcal{S})$, then by G_{19}, $\mathit{hf}(c)$.

$\underline{S_{12}[\mathcal{S},*.\mathit{find}](\{d=d^*\})}$**:** $S_{10}[\mathcal{S},*.\mathit{find}](\{d=d^*\})$ follows from $S_8(d)$, the fact that $d.\mathit{find}\equiv d$, $S_1[\mathcal{S}](d^*,d)$, and $S_9[*.\mathit{find}](d)$. We now show $S_{11}[\mathcal{S},*.\mathit{find}](\{d=d^*\})$. To show $d^*\equiv\mathit{canon}_\chi(\mathcal{S}(d^*))$, note that since $d^*\equiv\mathit{canon}_\chi(\mathcal{S}(d))$, it follows from Lemma A.1 that $\mathcal{S}(d^*)\equiv d^*$. Using property 2 of *canon*, it also follows that $\mathit{canon}_\chi(d^*)\equiv d^*$. Thus, $d^*\equiv\mathit{canon}_\chi(\mathcal{S}(d^*))$. $\mathit{fr}(d=d^*)$ follows by $\mathit{fr}(d^*)$ and $d.\mathit{find}\equiv d$. Finally, $S_4[\mathcal{S},*.\mathit{find}](\{d=d^*\})$ follows from $S_4[\mathcal{S},*.\mathit{find}](d^*)$, and $d.\mathit{find}\equiv d$ and G_6.

We now consider the properties at line 376.

$\underline{G}$**:** By G at 98.

$\underline{P_\subseteq(\mathcal{F},\mathit{lhs}(\mathcal{S}))}$**:** By $P_=(\mathcal{S},*.\mathit{find})$ at 374 and $P_\subseteq(\mathcal{F},\mathit{lhs}(\mathcal{S}))$ at 98.

$\underline{P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]}$**:** By $P_1[\Phi,\mathcal{H},\mathcal{I},\mathcal{N}]$ at 98.

$\underline{\neg\mathcal{I}'}$**:** By $\neg\mathcal{I}'$ at 374.

$\underline{S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]}$**:** By $S_3[\mathcal{A},\mathcal{I},*.\mathit{find}]$ at 98.

$\underline{\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})}$**:** By $P_{=}(*.\mathit{find})$ at 374 and $\mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R})$ at 98.

$\underline{\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})}$**:** By $P_{=}(*.\mathit{find})$ at 374 and $\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R})$ at 98.

$\underline{\mathcal{I} \vee (d \in \mathcal{R})}$**:** By $\mathcal{I} \vee (\mathit{lhs}(\mathcal{E}) \subseteq \mathcal{R})$ at 98.

Line 378:

370. $G,\ P_{=}(\mathit{all}),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \mathit{hf}(d),\ S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}],\ S_8(d),\ S_9[*.\mathit{find}](d)$
371. `IF` $\neg\mathcal{I}$ `AND Find(`d`)` $\equiv d$ `THEN BEGIN`
...
376. $G,\ P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S})),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ \neg\mathcal{I}',\ S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),$ $\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R}),\ \mathcal{I} \vee (d \in \mathcal{R})$
377. `END`
378. $G,\ P_{\subseteq}(\mathcal{F}, \mathit{lhs}(\mathcal{S})),\ P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}],\ S_3[\mathcal{A}, \mathcal{I}, *.\mathit{find}],\ \mathcal{I} \vee (\mathcal{M} \subseteq \mathcal{R}),$ $\mathcal{I} \vee (\mathcal{R}' \subseteq \mathcal{R}),\ \mathcal{I}' \to \mathcal{I},\ \underline{S_{34}[\mathcal{I}, \mathcal{S}, *.\mathit{find}](d)}$

$\underline{S_{34}[\mathcal{I}, \mathcal{S}, *.\mathit{find}](d)}$**:** From 370: By the if-condition, $\mathcal{I} \vee (d.\mathit{find} \not\equiv d)$. From 376: by $\mathcal{I} \vee (d \in \mathcal{R})$.

RewriteHelper

Line 385:

383. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(t) \subseteq \mathcal{V},\ t' \equiv\ t,\ P_{23}(t),\ \mathit{fr}(t) \vee \mathit{hf}(t),\ S_{25}[\mathcal{S}, *.\mathit{find}](t)$
384. `IF` t `is a` χ`-leaf THEN BEGIN`
385. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(t) \subseteq \mathcal{V},\ t' \equiv\ t,\ P_{23}(t),\ \mathit{fr}(t) \vee \mathit{hf}(t),\ \underline{\neg S_8(t)},$ $S_{25}[\mathcal{S}, *.\mathit{find}](t)$

$\underline{\neg S_8(t)}$**:** By the if-condition.

Line 387:

385. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(t) \subseteq \mathcal{V},\ t' \equiv\ t,\ P_{23}(t),\ \mathit{fr}(t) \vee \mathit{hf}(t),\ \neg S_8(t),$ $S_{25}[\mathcal{S}, *.\mathit{find}](t)$
386. `IF` $\neg$`HasFind(`t`) OR` $t.\mathit{find} \equiv\ t$ `THEN BEGIN`
387. $G,\ P_{=}(\mathit{all}),\ \mathit{free}(t) \subseteq \mathcal{V},\ t' \equiv\ t,\ \mathit{fr}(t),\ P_{23}(t),\ \neg S_8(t),$ $\underline{S_{31}[\mathcal{S}, *.\mathit{find}](t)},\ \underline{S_{32}[\mathcal{S}](t)},\ \underline{S_{33}[\mathcal{S}](t, t')}$

$\underline{S_{31}[\mathcal{S}, *.find](t)}$**:** Since t is a χ-leaf by $\neg S_8(t)$, we simply must show $S_4[\mathcal{S}, *.find](t)$. Suppose $s \trianglelefteq t$ and $\neg hf(s)$. We must show that $s \equiv canon_\chi(\mathcal{S}(s))$. If $s \triangleleft t$, then this follows by $S_{25}[\mathcal{S}, *.find](t)$. Otherwise, $s \equiv t$. Since s is a χ-leaf and $\neg hf(s)$, it follows from G_{19} that $s \notin \mathcal{S}$, and thus $\mathcal{S}(s) \equiv s$. But by property 4 of *canon*, $canon_\chi(s) \equiv s$, so $s \equiv canon_\chi(\mathcal{S}(s))$.

$\underline{S_{32}[\mathcal{S}](t)}$**:** By $\neg S_8(t)$, t is a χ-leaf, so we simply must show $\mathcal{S}(t) \equiv t$. We showed this above for the case when $\neg hf(t)$. Otherwise, we know that $t.find \equiv t$. Thus, by G_{19}, $t \notin lhs(\mathcal{S})$, and thus $\mathcal{S}(t) \equiv t$.

$\underline{S_{33}[\mathcal{S}](t, t')}$**:** We know that $t \equiv t'$. As shown above, $\mathcal{S}(t) \equiv t$, so $canon_\chi(t) \equiv canon_\chi(\mathcal{S}(t))$.

Line 389:

```
387.      G, P_=(all), free(t) ⊆ V, t' ≡ t, fr(t), P_23(t), ¬S_8(t),
          S_31[S, *.find](t), S_32[S](t), S_33[S](t, t')
388.      RETURN t;
389.      G, P_=(all), free(retval) ⊆ V, T ∪ Φ |= t' ≃ retval, P_23(retval),
          S_30[*.find](retval), S_31[S, *.find](retval), S_32[S](retval), S_33[S](retval, t')
```

$\underline{S_{30}[*.find](retval)}$**:** Trivial since t is a χ-leaf by $\neg S_8(t)$ (and thus $\delta_\chi(t) \equiv \{t\}$) and $fr(t)$.

Line 393:

```
391.      G, P_=(all), free(t) ⊆ V, t' ≡ t, P_23(t), hf(t), S_25[S, *.find](t)
392.      t := Find(t);
393.      G, P_=(all), free(t) ⊆ V, t' ∼ t, P_23(t), fr(t), S_25[S, *.find](t)
...
237. G, hf(t)
238. Find(t)
...
250. END Find
251. G, P_=(all), retval.find ≡ retval, t ∼ retval
```

$\underline{G}$**:** By G at 251.

$\underline{P_=(all)}$**:** By $P_=(all)$ at 391 and 251.

$\underline{free(t) \subseteq \mathcal{V}}$: By $retval.find \equiv retval$ at 251, $retval \lhd \mathcal{F}$, so $free(retval) \subseteq \mathcal{V}$ by G_8.

$\underline{t' \sim t}$: By $t' \equiv t_{391}$ and $t \sim retval$ at 251.

$\underline{P_{23}(t)}$: By $retval.find \equiv retval$ and G_7 at 251.

$\underline{fr(t)}$: By $retval.find \equiv retval$ at 251.

$\underline{S_{25}[\mathcal{S}, *.find](t)}$: Trivial since $hf(s)$ for all $s \trianglelefteq t$ by $retval.find \equiv retval$ at 251 and G_6.

Line 395:

```
393.      G, P_=(all), free(t) ⊆ V, t' ~ t, P_23(t), fr(t), S_25[S, *.find](t)
394.      RETURN RewriteHelper(t*);
395.      G, P_=(all), free(retval) ⊆ V, T ∪ Φ |= t' ≃ retval, P_23(retval),
          S_30[*.find](retval), S_31[S, *.find](retval), S_32[S](retval), S_33[S](retval, t')
...
381. G, free(t) ⊆ V, P_23(t), fr(t) ∨ hf(t), S_25[S, *.find](t)
382. RewriteHelper(t)
...
410. END RewriteHelper
411. G, P_=(all), free(retval) ⊆ V, T ∪ Φ |= t' ≃ retval, P_23(retval),
     S_30[*.find](retval), S_31[S, *.find](retval), S_32[S](retval), S_33[S](retval, t')
```

$\underline{G}$: By G at 411.

$\underline{P_{=}(all)}$: By $P_{=}(all)$ at 393 and 411.

$\underline{free(retval) \subseteq \mathcal{V}}$: By $free(retval) \subseteq \mathcal{V}$ at 411.

$\underline{\mathcal{T} \cup \Phi \models t' \simeq retval}$: By $t' \sim_{393} t_{393}$ and G_2 at 393, $\mathcal{T} \cup \Phi_{393} \models t' = t_{393}$. Thus, by $\mathcal{T} \cup \Phi \models t' \simeq retval$ and $P_{=}(all)$ at 411, $\mathcal{T} \cup \Phi \models t' \simeq retval$.

$\underline{P_{23}(retval)}$: By $P_{23}(retval)$ at 411.

$\underline{S_{30}[*.find](retval)}$: By $S_{30}[*.find](retval)$ at 411.

$\underline{S_{31}[\mathcal{S}, *.find](retval)}$: By $S_{31}[\mathcal{S}, *.find](retval)$ at 411.

$\underline{S_{32}[\mathcal{S}](retval)}$: By $S_{32}[\mathcal{S}](retval)$ at 411.

$\underline{S_{33}[\mathcal{S}](retval, t')}$: We must show that $canon_\chi(retval) \equiv canon_\chi(\mathcal{S}(t'))$. First of all, by $S_{33}[\mathcal{S}](retval, t')$ at 411, $canon_\chi(retval) \equiv canon_\chi(\mathcal{S}(t_{393}))$. Thus, by property 1 of *canon*, $\mathcal{T}_\chi \models \gamma_\chi(retval = \mathcal{S}(t_{393}))$. It follows that $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(retval = t_{393})$. Then, by the fact that $t' \sim_{393} t_{393}$ and G_{15} at 411, we have that $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(t' = t_{393})$. Thus, $\mathcal{T}_\chi \cup \gamma_\chi(\mathcal{S}) \models \gamma_\chi(retval = t')$. By Proposition 2.1, $\mathcal{T}_\chi \models \gamma_\chi(\mathcal{S}(retval) = \mathcal{S}(t'))$. Now, note that by $S_{32}[\mathcal{S}](retval)$, $\mathcal{S}(retval) \equiv retval$, so we have $\mathcal{T}_\chi \models \gamma_\chi(retval = \mathcal{S}(t'))$, and thus, by property 1 of *canon*, $canon_\chi(retval) \equiv canon_\chi(\mathcal{S}(t'))$.

Line 399:

384. IF t is a χ-leaf THEN BEGIN
...
398. END ELSE BEGIN
399. G, $P_=(all)$, $free(t) \subseteq \mathcal{V}$, $t' \equiv t$, $P_{23}(t)$, $fr(t) \vee hf(t)$, $\underline{S_8(t)}$, $S_{25}[\mathcal{S}, *.find](t)$

$\underline{S_8(t)}$: By the if-condition.

Line 401:

399. G, $P_=(all)$, $free(t) \subseteq \mathcal{V}$, $t' \equiv t$, $P_{23}(t)$, $fr(t) \vee hf(t)$, $S_8(t)$, $S_{25}[\mathcal{S}, *.find](t)$
400. FOR k := 1 to $Arity(t)$ DO BEGIN
401. G, $P_=(all)$, $free(t) \subseteq \mathcal{V}$, $\mathcal{T} \cup \Phi \models t' \simeq t$, $P_{23}(t)$, $S_8(t)$, $\underline{S_{37}[*.find](t)}$, $\underline{S_{38}[*.find](t,k)}$, $\underline{S_{39}[\mathcal{S}, *.find](t,k)}$, $\underline{S_{40}[\mathcal{S}, *.find](t,k)}$, $\underline{S_{41}[\mathcal{S}](t,k)}$, $Op(t) = Op(t')$
402. $t[k]$:= RewriteHelper($t[k]$);
403. G, $P_=(all)$, $free(t) \subseteq \mathcal{V}$, $\mathcal{T} \cup \Phi \models t' \simeq t$, $P_{23}(t)$, $S_8(t)$, $S_{37}[*.find](t)$, $S_{38}[*.find](t, k+1)$, $S_{39}[\mathcal{S}, *.find](t, k+1)$, $S_{40}[\mathcal{S}, *.find](t, k+1)$, $S_{41}[\mathcal{S}](t, k+1)$, $Op(t) = Op(t')$
404. END

The transition from 403 is trivial, so we consider only the transition from 399.

$\underline{S_{37}[*.find](t)}$: If $hf(t)$, then $hf(c)$ for each child of t by G_6. Otherwise, $\neg hf(t)$ and thus $fr(t)$. But this means that each child must be find-reduced.

$\underline{S_{38}[*.find](t,k)}$: Trivial, since $k = 1$.

$\underline{S_{39}[\mathcal{S}, *.find](t,k)}$**:** Trivial, since $k = 1$.

$\underline{S_{40}[\mathcal{S}, *.find](t,k)}$**:** By $S_{25}[\mathcal{S}, *.find](t)$ and $t' \equiv t$ at 399.

$\underline{S_{41}[\mathcal{S}](t,k)}$**:** Trivial, since $k = 1$.

Line 403:

```
401.      G, P_=(all), free(t) ⊆ V, T ∪ Φ |= t' ≃ t, P_23(t), S_8(t), S_37[*.find](t),
          S_38[*.find](t,k), S_39[S,*.find](t,k), S_40[S,*.find](t,k), S_41[S](t,k),
          Op(t) = Op(t')
402.      t[k] := RewriteHelper(t[k]);
403.      G, P_=(all), free(t) ⊆ V, T ∪ Φ |= t' ≃ t, P_23(t), S_8(t), S_37[*.find](t),
          S_38[*.find](t,k+1), S_39[S,*.find](t,k+1), S_40[S,*.find](t,k+1),
          S_41[S](t,k+1), Op(t) = Op(t')
...
381. G, free(t) ⊆ V, P_23(t), fr(t) ∨ hf(t), S_25[S,*.find](t)
382. RewriteHelper(t)
...
410. END RewriteHelper
411. G, P_=(all), free(retval) ⊆ V, T ∪ Φ |= t' ≃ retval, P_23(retval),
     S_30[*.find](retval), S_31[S,*.find](retval), S_32[S](retval), S_33[S](retval,t')
```

We first consider the preconditions of `RewriteHelper`.

$\underline{free(\mathbf{t[k]}) \subseteq \mathcal{V}}$**:** By $free(t) \subseteq \mathcal{V}$ at 401.

$\underline{P_{23}(\mathbf{t[k]})}$**:** By $P_{23}(t)$ at 401.

$\underline{fr(\mathbf{t[k]}) \vee hf(\mathbf{t[k]})}$**:** By $S_{37}[*.find](t)$ at 401.

$\underline{S_{25}[\mathcal{S}, *.find](\mathbf{t[k]})}$**:** By $S_{40}[\mathcal{S}, *.find](t,k)$ at 401.

Now we consider the properties at line 403.

$\underline{G}$**:** By G at 411.

$\underline{P_=(all)}$**:** By $P_=(all)$ at 401 and 411.

$\underline{free(t) \subseteq \mathcal{V}}$**:** By $free(t) \subseteq \mathcal{V}$ at 401 and $P_=(\mathcal{V})$ and $free(retval) \subseteq \mathcal{V}$ at 411.

$\mathcal{T} \cup \Phi \models t' \simeq t$: We know that $\mathcal{T} \cup \Phi_{401} \models t' = t_{401}$. Also, $\mathcal{T} \cup \Phi \models t_{401}[k] = t[k]$. But by $P_{=}(all)$, $\Phi_{401} = \Phi$, so $\mathcal{T} \cup \Phi \models t' = t$.

$P_{23}(t)$: By $P_{23}(t)$ at 401, $P_{23}(retval)$ at 411.

$S_8(t)$: By $S_8(t)$ at 401 since the operator of t has not changed.

$S_{37}[*.find](t)$: Let c be a child of t. We must show $hf(c) \vee fr(c)$. If $c \not\equiv t[k]$, then this follows from $S_{37}[*.find](t)$ at 401 and $P_{=}(all)$ at 411. If $c \equiv t[k]$, then by $S_{30}[*.find](retval)$ at 411, $\forall\, d \in \delta_{\chi}(c).fr(d)$. Thus, by G_{18}, $fr(c)$.

$S_{38}[*.find](t, k+1)$: Suppose $1 \leq l < k+1$. If $l \neq k$, then we have $S_{30}[*.find](t[l])$ by $S_{38}[*.find](t, k)$ at 401 and $P_{=}(all)$ at 411. Otherwise, if $l = k$, we must show $S_{30}[*.find](t[k])$. But this follows by $S_{30}[*.find](retval)$ at 411.

$S_{39}[\mathcal{S}, *.find](t, k+1)$: Follows by property $S_{39}[\mathcal{S}, *.find](t, k)$ at 401, and by $P_{=}(all)$, $S_{31}[\mathcal{S}, *.find](retval)$, and $S_{32}[\mathcal{S}](retval)$ at 411.

$S_{40}[\mathcal{S}, *.find](t, k+1)$: By $S_{40}[\mathcal{S}, *.find](t, k)$ at 401 and $P_{=}(all)$ at 411.

$S_{41}[\mathcal{S}](t, k+1)$: Suppose $1 \leq l < k+1$. If $l \neq k$, then $S_{33}[\mathcal{S}](t[l], t'[l])$ by $S_{41}[\mathcal{S}](t, k)$ at 401 and $P_{=}(all)$ at 411. Otherwise, by $S_{40}[\mathcal{S}, *.find](t, k)$ at 401, $t_{401}[l] \equiv t'[l]$, and thus, by $S_{33}[\mathcal{S}](retval, t')$ at 411, $S_{33}[\mathcal{S}](t[l], t'[l])$.

$Op(t) = Op(t')$: By $Op(t) = Op(t')$ at 401.

Line 405:

399. $G,\ P_{=}(all),\ free(t) \subseteq \mathcal{V},\ t' \equiv\ t,\ P_{23}(t),\ fr(t) \vee hf(t),\ S_8(t),$
$S_{25}[\mathcal{S}, *.find](t)$

400. `FOR` k `:= 1 to` $Arity(t)$ `DO BEGIN`

...

403. $\underline{G,\ P_{=}(all),\ free(t) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models t' \simeq t,\ P_{23}(t),\ S_8(t),\ S_{37}[*.find](t),}$
$\underline{S_{38}[*.find](t, k+1),\ S_{39}[\mathcal{S}, *.find](t, k+1),\ S_{40}[\mathcal{S}, *.find](t, k+1),}$
$\underline{S_{41}[\mathcal{S}](t, k+1),\ Op(t) = Op(t')}$

404. `END`

405. $G,\ P_{=}(all),\ free(t) \subseteq \mathcal{V},\ \mathcal{T} \cup \Phi \models t' \simeq t,\ P_{23}(t),\ \underline{S_{30}[*.find](t)},$
$\underline{S_{31}[\mathcal{S}, *.find](t)},\ \underline{S_{32}[\mathcal{S}](t)},\ \underline{S_{33}[\mathcal{S}](t, t')}$

Note that no transition from 399 is possible: t is a compound term by $S_8(t)$, so it must have at least one child.

$\underline{S_{30}[*.find](t)}$**:** By $S_8(t)$, $t \notin \delta_\chi(t)$, so $S_{30}[*.find](t)$ follows from $S_{38}[*.find](t, k+1)$ at 403 and the end-of-loop condition, $k = Arity(t)$.

$\underline{S_{31}[\mathcal{S}, *.find](t)}$**:** As above, we have by $S_8(t)$ that $t \notin \delta_\chi(t)$, so $S_{31}[\mathcal{S}, *.find](t)$ follows from $S_{39}[\mathcal{S}, *.find](t, k+1)$ at 403 and the end-of-loop condition.

$\underline{S_{32}[\mathcal{S}](t)}$**:** By $t \notin \delta_\chi(t)$, $S_{39}[\mathcal{S}, *.find](t, k+1)$ at 403, and the end-of-loop condition.

$\underline{S_{33}[\mathcal{S}](t, t')}$**:** We must show $canon_\chi(t) \equiv canon_\chi(\mathcal{S}(t'))$. Let $n = Arity(t)$ and let $f = Op(t)$. Note that $f = Op(t')$ as well. The proof is as follows:

$canon_\chi(t)$		
$\equiv$	$canon_\chi(f(t[1], \ldots, t[n])$	def. of t
$\equiv$	$canon_\chi(f(canon_\chi(t[1]), \ldots, canon_\chi(t[n])))$	Lemma A.3
$\equiv$	$canon_\chi(f(canon_\chi(\mathcal{S}(t'[1])), \ldots, canon_\chi(\mathcal{S}(t'[n]))))$	$S_{41}[\mathcal{S}](t, k+1)$
$\equiv$	$canon_\chi(f(\mathcal{S}(t'[1]), \ldots, \mathcal{S}(t'[n])))$	Lemma A.3
$\equiv$	$canon_\chi(\mathcal{S}(f(t'[1], \ldots, t'[n])))$	$\mathcal{T}(f) = \chi$
$\equiv$	$canon_\chi(\mathcal{S}(t'))$	def. of t'

A.6 Partial Correctness

As stated before, partial correctness means that if the program terminates, it gives the correct result. For a validity checker, partial correctness can further be divided into soundness and completeness. In order to show that the framework guarantees soundness and completeness, we must first show that the preconditions of `AddFact` are always satisfied. We then show that the postconditions of `AddFact` guarantee soundness and completeness.

A.6.1 Preconditions of AddFact

We first consider the initial call to `AddFact`. Given the initial values of all the global state variables, it is not hard to see that initially, all preconditions of `AddFact` are trivially true. The only one which does not follow immediately is $P_2(e)$ which requires that the parameter to `AddFact` be a Σ-formula. Clearly, we expect this precondition to be satisfied by any user of the framework.

Now, consider subsequent calls to `AddFact`. As long as the user code does not change any of the global state of the framework, every precondition of `AddFact` is guaranteed by the postconditions of `AddFact` except for $P_3[\Phi, \mathcal{H}](e)$ and $P_2(e)$. As mentioned, $P_2(e)$ is a reasonable expectation for any call to `AddFact`. The other case, $P_3[\Phi, \mathcal{H}](e)$ is more interesting. Basically, $P_3[\Phi, \mathcal{H}](e)$ requires that free variables in the formula passed to `AddFact` either be free variables appearing in the assumption history $\mathcal{H}$ (i.e. in previous calls to `AddFact`) or fresh variables not being used by the framework. This seems reasonable, since it is easy to enforce that the framework and the user code generate different sets of fresh variables.

However, there is a potential difficulty with this precondition. If the user code uses the `Simplify` procedure to simplify formulas, the two sets of fresh variables may get mixed up. It seems reasonable for the user code to use parts of formulas returned to it by `Simplify` to construct new facts to pass to `AddFact`. This may violate the precondition.

This problem can be overcome by redefining $\mathcal{H}$ as the value of Φ at the beginning of a call to `AddFact` and removing the precondition $P_3[\Phi, \mathcal{H}](e)$. Essentially, what

this does is change the functionality of `AddFact`. Instead of collecting a set of facts in $\mathcal{H}$, each call to `AddFact` transforms the current fact database Φ into a new fact database which is equisatisfiable with $\Phi \cup \{e\}$.

A.6.2 Soundness

At a high level, a validity checker such as CVC is sound if, whenever it reports that a formula is valid, the formula really is valid. This can be guaranteed if we know that whenever the framework sets $\mathcal{I}$ to be `TRUE`, the assumption history $\mathcal{H}$ is truly inconsistent.

Theorem A.1. *If a call to* `AddFact` *respects the preconditions and* `AddFact` *terminates with* $\mathcal{I}$ *set to* `TRUE`*, then* $\mathcal{T} \cup \mathcal{H} \models \mathit{false}$.

Proof. $\mathcal{I} \rightarrow (\mathcal{T} \cup \mathcal{H} \models \mathit{false})$ is the first global invariant. Thus, soundness is ensured by the fact that G_1 is a postcondition of `AddFact`. □

A.6.3 Completeness

A validity checker is complete if, whenever a valid formula is provided as input, the validity checker reports that the formula is valid. This can be guaranteed as long as we know that whenever $\mathcal{I}$ is `FALSE`, the assumption history $\mathcal{H}$ is actually satisfiable.

In reality, because we are allowing non-convex theories (see Section 3.5.1), we can't prove something quite this strong. Instead, we can show that whenever $\mathcal{I}$ is `FALSE` and *convex* is true, the assumption history is satisfiable. With the additional assumption that every branch of the decision tree eventually reaches a point at which *convex* holds, it follows that CVC is complete. We now give a proof of this based on postconditions of `AddFact`.

Theorem A.2. *If a call to* `AddFact` *respects the preconditions,* `AddFact` *terminates with* $\mathcal{I} =$ `FALSE`*, and convex is true, then* $\mathcal{T} \cup \mathcal{H}$ *is satisfiable.*

Proof. By $P_{10}[\Phi, \mathcal{I}, \mathcal{N}, \Lambda_*, *.\mathit{find}]$, we have that $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$ is satisfiable for each theory $\mathcal{T}_i$. We first show that this implies that $\mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\approx})$ is satisfiable

for each $\mathcal{T}_i$, where $\approx$ is the restriction of $\sim$ to Λ (recall that $\Lambda = \bigcup \Lambda_i$). Suppose $M \models_\rho \mathcal{T}_i \cup \gamma_i(\Phi_i \cup Ar_{\sim_i})$, and consider the difference in the domains of $Ar_{\sim_i}$ and $Ar_\approx$. The domain of $Ar_\approx$ includes all terms in Λ, while $Ar_{\sim_i}$ includes only the terms in Λ_i. Thus, if t is in the domain of $Ar_\approx$ but not the domain of $Ar_{\sim_i}$, then it must be the case that $t \notin \Lambda_i$ and $t \in \Lambda_j$ where $j \neq i$. Now, if $\mathcal{T}(t) = i$, then by G_{10} and G_9, it follows that $t \in \Lambda_i$, so we must have $\mathcal{T}(t) \neq i$. Similarly, if t occurs i-alien in some formula $e \in \Phi_i$, then by G_9, $t \in \Lambda_i$. Thus, $\mathcal{T}(t) \neq i$ and t does not occur i-alien in any formula in Φ_i. It follows that $\gamma_i(t)$ is a variable and does not appear in $\gamma_i(\Phi_i)$. Now, we can modify ρ so that it also satisfies $\gamma_i(Ar_\approx)$. To do so, we simply associate a different element of M with each equivalence class of $\sim$ as follows: if the equivalence class contains a term t such that $\gamma_i(t)$ appears in $\gamma_i(\Phi_i \cup Ar_{\sim_i})$, then we associate the element assigned to this term by M and ρ. Otherwise, we associate a new element of M (we can assume M has infinitely many elements because $\mathcal{T}_i$ is stably infinite). Then, for each term t such that t is in the domain of $Ar_\approx$ but not in the domain of $Ar_{\sim_i}$, we modify ρ to assign to $\gamma_i(t)$ the element associated with the equivalence class of t. Call the modified assignment ρ'. Since, as shown above, $\gamma_i(t)$ does not appear in $\gamma_i(\Phi_i \cup Ar_{\sim_i})$, it is not hard to see that $M \models_{\rho'} \gamma_i(\Phi_i \cup Ar_\approx)$.

Now, since $\gamma_i(\Phi_i \cup Ar_\approx)$ is satisfiable in $\mathcal{T}_i$ for each i, it follows by Theorem 2.3 that $\mathcal{T} \cup \Phi$ is satisfiable. By $P_1[\Phi, \mathcal{H}, \mathcal{I}, \mathcal{N}]$, $\mathcal{T} \cup \mathcal{N} \cup \Phi \models \mathcal{H}$. But because *convex* holds, it follows that $\mathcal{T} \cup \Phi \models \mathcal{N}$, so $\mathcal{T} \cup \Phi \models \mathcal{H}$ and thus $\mathcal{T} \cup \mathcal{H}$ is satisfiable. □

Bibliography

[1] C. Barrett, D. Dill, and J. Levitt. Validity Checking for Combinations of Theories with Equality. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods In Computer-Aided Design*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, November 1996. Palo Alto, California.

[2] C. Barrett, D. Dill, and A. Stump. A Framework for Cooperating Decision Procedures. In David McAllester, editor, *17th International Conference on Computer-Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–97. Springer-Verlag, June 2000. Pittsburgh, Pennsylvania.

[3] Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. A Decision Procedure for Bit-Vector Arithmetic. In *Proceedings of the 35th Design Automation Conference*, June 1998. San Francisco, CA.

[4] Ritwik Bhattacharya. Private communication, 1999.

[5] Nikolaj S. Bjørner. *Integrating Decision Procedures for Temporal Verification.* PhD thesis, Stanford University, 1999.

[6] Vamsi Boppana, Sreeranga P. Rajan, Koichiro Takayama, and Masahiro Fujita. Model Checking Based on Sequential ATPG. In *11th International Conference on Computer-Aided Verification*, pages 418–430. Springer-Verlag, July 1999. Trento, Italy.

[7] R. Bryant, S. German, and M. Velev. Exploiting Positive Equality in a Logic of Equality with Uninterpreted Functions. In *11th International Conference on*

Computer-Aided Verification, volume 1633 of *Lecture Notes in Computer Science*, pages 470–482. Springer-Verlag, July 1999. Trento, Italy.

[8] Tevfik Bultan, Richard Gerber, and William Pugh. Symbolic Model Checking of Infinite State Systems Using Presburger Arithmetic. In Orna Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 400–411. Springer-Verlag, June 1997. Haifa, Israel.

[9] Jerry R. Burch and David L. Dill. Automatic Verification of Pipelined Microprocessor Control. In David L. Dill, editor, *Conference on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 68–80. Springer-Verlag, 1994. Stanford, California, June 21–23, 1994.

[10] D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's Decision Procedure for Combinations of Theories. In M. McRobbie and J. Slaney, editors, *13th International Conference on Computer Aided Deduction*, volume 1104 of *Lecture Notes in Computer Science*, pages 463–477. Springer-Verlag, 1996.

[11] Satyaki Das and David L. Dill. Successive Approximation of Abstract Transition Relations. In *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science*, 2001. June 2001, Boston, USA.

[12] Satyaki Das, David L. Dill, and Seungjoon Park. Experience with Predicate Abstraction. In *11th International Conference on Computer-Aided Verification*, pages 160–172. Springer-Verlag, July 1999. Trento, Italy.

[13] Nancy A. Day, John Launchbury, and Jeff Lewis. Logical Abstractions in Haskell. In *Proceedings of the 1999 Haskell Workshop*. Utrecht University Department of Computer Science, Technical Report UU-CS-1999-28, October 1999.

[14] Leonardo de Moura, Harald Ruess, and Maria Sorea. Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In *18th International Conference on Automated Deduction*, 2002.

[15] L. e Silva, L. Silveira, and J. Marques-Silva. Algorithms for Solving Boolean Satisfiability in Combinational Circuits. In *Proceedings of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, March 1999.

[16] H. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[17] Z. Manna et al. STeP: Deductive-Algorithmic Verification of Reactive and Real-time Systems. In *8th International Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418. Springer-Verlag, 1996.

[18] C. Flanagan. Private Communication, 2000.

[19] Cormac Flanagan, Rajeev Joshi, and James B. Saxe. The Design of An Efficient Theorem Prover using Explicated Clauses, 2002. In Preparation.

[20] M. Gordon and T. Melham. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[21] Søren T. Heilmann. *Proof Support for Duration Calculus*. PhD thesis, Technical University of Denmark, 1999.

[22] Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.

[23] Jeremy Levitt and Kunle Olukotun. A Scalable Formal Verification Methodology for Pipelined Microprocessors. In ACM-SIGDA; IEEE, editor, *Proceedings of the 33th ACM/IEEE Design Automation Conference*, pages 558–563, Las Vegas, NV, June 1996. ACM Press.

[24] Jeremy Levitt and Kunle Olukotun. Verifying Correct Pipeline Implentation for Microprocessors. In *International Conference on Computer Aided Design*, San Jose, CA, November 1997. IEEE Computer Society Press.

[25] R. B. Jones, D. L. Dill, and J. R. Burch. Efficient Validity Checking for Processor Verification. In *IEEE/ACM International Conference on Computer Aided Design*, pages 2–6, November 1995.

[26] Tracy Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):4–15, January 1992.

[27] J. Levitt. *Formal Verification Techniques for Digital Systems.* PhD thesis, Stanford University, 1999.

[28] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 39th Design Automation Conference*, June 2001. Las Vegas, NV.

[29] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.

[30] Derek C. Oppen. Complexity, Convexity and Combinations of Theories. *Theoretical Computer Science*, 12:291–302, 1980.

[31] S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, 1992.

[32] David Y.W. Park, Jens U. Skakkebæk, Mats P.E. Heimdahl, Barbara J. Czerny, and David L. Dill. Checking Properties of Safety Critical Specifications Using Efficient Decision Procedures. In *FMSP'98: Second Workshop on Formal Methods in Software Practice*, pages 34–43, March 1998.

[33] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding Equality Formulas by Small-Domain Instantiations. In *11th International Conference on Computer-Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 455–469. Springer-Verlag, July 1999. Trento, Italy.

[34] H. Ruess and N. Shankar. Deconstructing Shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, June 2001.

[35] R. Shostak. Deciding combinations of theories. *Journal of the Association for Computing Machinery*, 31(1):1–12, 1984.

[36] Laurent Simon. The Sat-Ex Site. http://www.lri.fr/∼simon/satex/satex.php3.

[37] A. Stump, C. Barrett, and D. Dill. CVC: a Cooperating Validity Checker. In *14th International Conference on Computer-Aided Verification*, 2002.

[38] Aaron Stump, David L. Dill, Clark W. Barrett, and Jeremy Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *16th IEEE Symposium on Logic in Computer Science*, pages 29–37. IEEE Computer Society, June 2001. Boston, Massachusetts.

[39] J. Su, D. Dill, and J. Skakkebæk. Formally Verifying Data and Control with Weak Reachability Invariants. In *Formal Method In Computer-Aided Design*, 1998.

[40] C. Tinelli and M. Harandi. A New Correctness Proof of the Nelson-Oppen Combination Procedure. In F. Baader and K. Schulz, editors, *1st International Workshop on Frontiers of Combining Systems (FroCoS'96)*, volume 3 of *Applied Logic Series*. Kluwer Academic Publishers, 1996.

[41] Cesare Tinelli and Christophe Ringeissen. Unions of Non-Disjoint Theories and Combinations of Satisfiability Procedures. Technical Report 01-02, Department of Computer Science, University of Iowa, April 2001.

[42] A. Tiwari. *Decision Procedures in Automated Deduction*. PhD thesis, State University of New York at Stony Brook, 2000.