DRAFT -- submitted to TOPLAS 7/99

Theory Generation for Security Protocols

Darrell Kindred Jeannette M. Wing

Computer Science Department
Carnegie Mellon University
5000 Forbes Ave.
Pitsburgh, PA 15213
{dkindred,wing}@cs.cmu.edu

July 16, 1999

Abstract

We introduce *theory generation*, a new general-purpose technique for performing automated verification. Theory generation draws inspiration from, and complements, both automated theorem proving and symbolic model checking, the two approaches that currently dominate mechanical reasoning. At the core of this approach is the notion of producing a finite representation of a theory—all the facts derivable from a set of assumptions. We present an algorithm for producing compact theory representations for an expressive class of simple logics.

Security-sensitive protocols are widely used today, and the growing popularity of electronic commerce is leading to increasing reliance on them. Though simple in structure, these protocols are notoriously difficult to design properly. Since specifications of these protocols typically involve only a small number of principals, keys, nonces, and messages, and since many properties of interest can be expressed in "little logics" such as the Burrows, Abadi, and Needham (BAN) logic of authentication, this domain is amenable to theory generation.

Theory generation enables fast, automated analysis of these security protocols. Given the theory representation generated from a protocol specification, one can quickly test for specific desired properties, as well as directly manipulate the representation to perform other kinds of analysis, such as protocol comparison. This paper describes applications of theory generation to more than a dozen security protocols using four different logics of belief; these examples confirm, or in some cases expose flaws in earlier analyses.

1 Introduction

1.1 Motivation

Security-sensitive protocols are widely used today, and we will rely on them even more heavily as electronic commerce continues to expand. This class of protocols includes well-known authentication protocols such as Kerberos [MNSS87] and Needham-Schroeder [NS78], newer protocols for electronic commerce such as NetBill [ST95] and Secure Electronic Transactions (SET) [VM96], and "security-enhanced" versions of existing network protocols, such as Netscape's Secure Sockets Layer (SSL) [FKK96], Secure HTTP [RS96], and Secure Shell (SSH) [Gro99].

These protocols are notoriously difficult to design properly. Researchers have uncovered critical but subtle flaws in protocols that had been scrutinized for years or even decades. Protocols that were secure in the environments for which they were designed have been used in new environments where their assumptions fail to hold, with dire consequences. These assumptions are often implicit and easily overlooked. Furthermore, security protocols by their nature demand a higher level of assurance than many systems and programs, since the use of these protocols implies that the user perceives a threat from malicious parties. The weakest-link argument requires that every component of a system be secure; since almost every modern distributed system makes use of some of these protocols, their security is crucial.

Given these considerations, we must apply careful reasoning to gain confidence in security protocols. In current practice, these protocols are analyzed sometimes using formal methods based on security-related logics such as the Burrows, Abadi, and Needham (BAN) logic of authentication, and sometimes using informal arguments and public review. While informal approaches play an important role, formal methods offer hope for producing more convincing evidence that a protocol meets its requirements. It is for critical system properties like security that the cost of applying formal methods can most easily be justified. The process of encoding protocols and security properties in a general-purpose verification system is often cumbersome and error-prone, and it sometimes requires that the protocol be expressed in an unnatural way. As a result, the cost of applying formal reasoning may be seen as prohibitive and the benefits uncertain; thus, protocols are often used with only informal or possibly-flawed formal arguments for their soundness. In recent years, there has been much interest in more "lightweight" verification approaches; the developers of domain-specialized model checkers have sought to satisfy this need. If we can develop formal methods that demand less from the user while still providing strong assurances, the result should be more dependable protocols and systems. In this paper, we offer what is essentially a special-purpose theorem prover. It is fast and automatic, in return for its restricted applicability.

1.2 Overview of Approach

We introduce a new technique, *theory generation*, which can be used to analyze these protocols and facilitate their development. This approach provides fully automated verification of the properties of interest, and feedback on the effects of refinements and modifications to protocols.

At the core of this approach is the notion of producing a finite representation of all the facts derivable from a protocol specification. The common protocols and logics in this domain have some special properties that make this approach appealing. First, the protocols can usually be expressed in terms of a small, finite number of participants, keys, messages, nonces, and so forth. Second, the logics with which we reason about them often comprise a finite number of rules of inference that cause "growth" in a controlled manner. The BAN logic of authentication, along with some other logics of belief and knowledge, meets this criterion. Together, these features of the domain make it practical to produce such a finite representation quickly and automatically.

The finite representation takes the form of a set of formulas T, which is essentially the transitive closure of the formulas constituting the protocol specification, over the rules of inference in a logic. This set is called the *theory*, or *consequence closure*. Given such a representation, verifying a specific property of interest, ϕ, requires a simple membership test: $\phi \in T$. In practice, the representation is not the entire transitive closure, and the test is slightly more involved than simple membership, but it is similar in spirit. Beyond this traditional property-testing form of verification, we can make further uses of the set T, for instance in comparing different versions of a protocol. We can capture some of the significant differences between protocols P and Q by examining the formulas that lie in the set difference $T_P \setminus T_Q$ and those that lie in $T_Q \setminus T_P$.

Using this new approach, we can provide protocol designers with a powerful and automatic tool for analyzing protocols while allowing them to express the protocols in a natural way. In addition, the tool can be instantiated with a simple representation of a logic, enabling the development of logics tailored to the verification task at hand without sacrificing push-button operation.

Beyond the domain of cryptographic protocols, theory generation could be applied to reasoning with any logic that exhibits the sort of controlled growth mentioned above (and explained formally in Section 2). For instance, researchers in artificial intelligence often use such logics to represent planning tasks, as in the Prodigy system [VCP+95].

The new approach makes it easy to generate automatically a checker specialized to a given logic. Just as Jon Bentley has argued the need for "little languages" [Ben86], this generator provides a way to construct "little checkers" for "little logics." We built such a checker generator tool called REVERE. The generated checkers are lightweight and quick, just as the logics are little in the sense of having limited connectives and restricted rules, but the results can be illuminating. Using REVERE we built four checkers based on three previously published belief logics and one new one, RV, which we derived from BAN. Using our checkers we analyzed over a dozen classic authentication protocols and a sampling of simplified electronic commerce protocols.

1.3 Road Map

In the remainder of this paper, we describe the theory and practice of theory generation for security protocol verification. In Section 2 we describe requirements for representing theories compactly; in Section 3, we present an algorithm for performing theory generation of these representations; and in Section 4, we give proofs of correctness and termination of the algorithm. Section 5 presents theory generation as applied using four different belief logics for analyzing well-known security protocols. In Section 6 we discuss how theory generation relates to other mechanized verification approaches. Finally, in Section 7 we summarize the contributions of our work and reflect on directions for future research.

2 Theory Generation Representation

When working with a logic, whether for program verification, planning, diagnosis, or any other purpose, a natural question to ask is, "What conclusions can we derive from our assumptions?" Given a logic, and some set of assumptions, T^0, we consider the complete theory, T^*, induced by T^0. Also known as the *consequence closure*, T^* is simply the (possibly infinite) set of formulas that can be derived from T^0, using the rules and axioms of the logic. We typically explore T^* by probing it at specific points: "Can we derive formula F? What about G?" Sometimes we test whether T^* contains every formula; that is, whether the theory is inconsistent. It is interesting, however, to consider whether we can characterize T^* more directly, and if so what benefits that might bring. In this section, we explore for a special class of logics a general way of representing T^*, and in the next section, we present a technique, called *theory generation*, for mechanically producing that representation.

2.1 Logic

Theory generation may in principle be applied to any logic, but in this paper we consider only those falling within a simple fragment of first-order logic. We introduce a class of "little logics" that are parameterized by sets of rules and rewrites. In this context, *rules* are universally quantified Horn clauses (e.g., $f(X,Y) \wedge g(X) \Rightarrow h(Y)$), and *rewrites* are universally quantified term equalities (e.g., $f(X,Y) = f(Y,X)$).

Definition 1 *The logic, ℓ_{RW}, where R is a set of rules, and W is a set of rewrites, has as its formulas all connective-free formulas of first order logic. The rules of inference of ℓ_{RW} are all the rules in R, as well as* *instantiation, and substitution of equal terms using the rewrites in W.*

Proofs in ℓ_{RW} are thus finite sequences of formulas in which each formula is either an assumption, an instance of an earlier formula, the result of applying one of the rules in R, or the result of a replacement using a rewrite in W.

We now define the notion of *theory* (also known as *consequence closure*):

Definition 2 *Let L be a logic, and T^0 a set of wffs in that logic. T^*, the* theory induced by T^0, *is the possibly infinite set of wffs containing exactly those wffs that can be derived from T^0 and the axioms of L, using the rules of L.*

T^* is closed with respect to inference, in that every formula that can be derived from T^* is a member of T^*. In the remainder of this section and in Section 3, we consider theories in the context of ℓ_{RW} logics and show how to generate representations of those theories.

2.2 Theory Representation

The goal of theory generation is to produce a finite representation of the theory induced by some set of assumptions; in this section we consider the forms this representation might take, and the factors that may weigh in favor of one representation or another. These factors will be determined in part by the purposes for which we plan to use the representation. There are three primary uses for the generated theory representation:

- It may be used in a decision procedure for determining the derivability of specific formulas of interest.
- It may be manipulated and examined by a human through assorted filters, in order to gain insight into the nature of the complete theory.
- It may be directly compared with the representation of some other theory, to reveal differences between the two theories.

Since the full theory is a possibly infinite set of formulas entailed by some initial assumptions, the clearest requirement of the representation we generate is that it be finite. We can achieve this requirement by selecting a finite set of "representative formulas" that belong to the theory, and let this set represent the full theory. Other approaches are conceivable; we could construct a notation for expressing certain infinite sets of formulas, perhaps analogous to regular expressions or context-free grammars. However, the logic itself is already quite expressive; indeed, the set of initial assumptions and rules "expresses" the full theory in some sense. There is no clear benefit of creating a separate language for theory description.

Given that we choose to represent a theory by selecting some subset of its formulas, it remains to decide what subset is best. Here are a few informal criteria that may influence our choice:

C1. The set of formulas must be finite, and should be small enough to make direct manipulation practical. An enormous, though finite, set would probably be not only inefficient to generate, but unsuitable for examination by a human except through very fine filters.

C2. There should be an algorithm that generates the set with reasonable efficiency.

C3. Given an already-generated set, there should be an efficient decision procedure for the full theory, using that set. Since the simplest way to characterize a theory is to test specific formulas for membership, a quick decision procedure is important.

C4. The set should be canonical. For a given set of initial assumptions, the generated theory representation should be uniquely determined. This makes direct comparison of the sets more useful.

C5. The set should include as many of the "interesting" formulas in the theory as possible, and as few of the "uninteresting" ones as possible. For instance, when a large class of formulas exists in the theory, but all represent essentially the same fact, it might be best for the theory representation to include only the simplest formula from this class. This will enable humans to glean useful information from the generated set without sifting through too much chaff.

Ganzinger, Nivela, and Niewenhuis's SATURATE prover takes one approach to this problem [NN93]. SATURATE is designed to work with a very general logic: full first-order logic over transitive relations. It can, under some circumstances, produce a finite "saturated theory" that enables an efficient decision procedure. The saturation process can also be used to check the consistency of a set of formulas, since **false** will appear in the saturated set if the original set is inconsistent. The price SATURATE pays for its generality is that saturation is not guaranteed to terminate in all cases; there are a number of user-tunable parameters that control the saturation process and make it more or less thorough and more or less likely to terminate. This flexibility is sometimes necessary, but we choose to focus on more limited logics in which we can make stronger guarantees regarding termination and require less assistance from the user.

For ℓ_{RW} logics, we select a class of theory representations we refer to as (R, R') *representations*:

Definition 3 *From the set of rules, R, choose some subset, R'. An (R, R')* representation *of the theory induced by T^0 contains a set of formulas derivable from T^0, such that any proof from the assumptions, T^0, in which the last rule application (if any) is an application of some rule in R', the conclusion of that proof is equivalent to some formula in the set. Furthermore, every formula in the set is equivalent to the conclusion of some such proof. Finally, no two formulas in the set are equivalent.*

In this formulation, the rules in R' are "preferred" in that they are applied as far as possible. The equivalence used in this definition may be strict identity or some looser equivalence relation. We can test a formula for membership in the full theory by using the theory representation and just the rules in $(R \setminus R')$ (we prove this in Section 4). This test can be significantly more efficient than the general decision problem using all of R, so criterion C3 can be satisfied. If we select the representation to include only one canonical formula from any equivalence class, then the representation is completely determined by T^0, R, and R', so criterion C4 is satisfied.

In order to satisfy the remaining criteria (C1, C2, and C5), we must choose R' carefully. We could choose $R' = \{\}$, but the resulting theory representation would just be T^0, the initial assumptions—unlikely to enable an efficient decision procedure and certainly not very "interesting" (in the sense of C5). For some sets of rules, we could let $R' = R$, but in many cases this would yield an infinite representative set, violating C1. In the next section we describe a method for selecting R' that is guaranteed to produce a finite representative set, and that satisfies the remaining criteria in practice.

3 The Theory Generation Algorithm, TG_ℓ

In an ℓ_{RW} logic, as defined in Section 2.1, we can automatically generate a finite representation of the theory induced by some set of formulas, Γ, provided the logic and the formulas in Γ satisfy some additional restrictions. In the following section, we describe these preconditions and explain how they can be checked. Sections 3.2 and 3.3 contain a description the algorithm itself, which produces an (R, R') theory representation given assumptions, rules, and rewrites satisfying the stated preconditions. Arguments for the correctness and termination of the algorithm appear in Sections 4.1 and 4.2, with full proofs in Appendix A.

Finally, in Section 3.4 we present an efficient decision procedure for *mostly-ground* formulas of ℓ_{RW}, which makes use of the theory representation.

3.1 Preconditions

In order to apply our theory generation algorithm, TG_ℓ, to a logic, ℓ_{RW}, and a set of assumptions, Γ, some preconditions on the rules and rewrites (R and W) of ℓ_{RW}, and on Γ, must hold. These preconditions require that the set of rules (R) can be partitioned into *S-rules* and *G-rules*, that the rewrites (W) be *size-preserving*, and that the formulas in Γ be *mostly-ground*. Informally, the S-rules are "shrinking rules," since they tend to produce conclusions no larger than their premises, and the G-rules are "growing rules" since they have the opposite effect. The S-rules are the principal rules of inference; in the generated theory representation, they will be treated as the preferred rules (the R' set in an (R, R') representation). We define these terms and formally present the preconditions in this section.

The TG_ℓ algorithm repeatedly applies rules, starting from the assumptions, to produce an expanding set of derivable formulas. The basic intent of these preconditions is to limit the ways in which formulas can "grow" through the application of rules. As long as the process of applying rules cannot produce formulas larger than the ones we started with, we can hope to reach a fixed point, where no further application of the rules can yield a new formula. The algorithm eagerly applies the S-rules to the assumptions as far as possible, using the G-rules and rewrites only as necessary. The restrictions below ensure first that the algorithm can find a new S-rule application in a finite number of steps, and second that each new formula derived is smaller than some already-known formula, so the whole process will terminate.

The preconditions below are defined with respect to a pre-order (a reflexive and transitive relation) on terms and formulas, $\preceq$. This pre-order may be defined differently for each application of the algorithm, but it must always satisfy these conditions, for all ℓ_{RW} formulas, F, G, T, T_1, and T_2, and all variables, X:

P1. The pre-order must be monotonic; that is,

$$(T_1 \preceq T_2) \Rightarrow ([T_1 \backslash X]F \preceq [T_2 \backslash X]F)$$

P2. The pre-order must be preserved under substitution:

$$(F \preceq G) \Rightarrow ([T \backslash X]F \preceq [T \backslash X]G)$$

P3. The set $\{F \mid F \preceq G\}$ must be finite (modulo variable renaming) for all formulas G. This is a form of well-foundedness.

Intuitively, $F \preceq G$ means that the formula F is no larger than G; with this interpretation, condition P3 means that there are only finitely many formulas no larger than G. In Section 5, we describe a specific $\preceq$ relation satisfying these conditions, which can be used with only minor variations in a variety of situations.

If there exists some such $\preceq$ under which the preconditions below are met, then the TG_ℓ algorithm can be applied to ℓ_{RW} and Γ.

We can weaken the condition P3 above slightly, to allow broader application of the algorithm. We introduce a syntactic constraint on formulas, represented as a set of formulas, $\mathcal{C}$. Every assumption in Γ must be in $\mathcal{C}$, and $\mathcal{C}$ must be closed over all the rules and rewrites (that is, when applied to formulas in $\mathcal{C}$, rules and rewrites must yield conclusions in $\mathcal{C}$). Furthermore, applying a substitution to a formula in $\mathcal{C}$ must always yield another formula in $\mathcal{C}$. We can then allow a pre-order $\preceq$ for which P3 above may not hold but P3$'$ does:

P3$'$. The set $\{F \in \mathcal{C} \mid F \preceq G\}$ must be finite (modulo variable renaming) for all formulas G.

In describing the preconditions, we use a notion of *mostly-ground* formulas:

Definition 4 *A formula, F, is* mostly-ground *with respect to a pre-order, $\preceq$, and some syntactic constraint, $\mathcal{C}$, if*

$$\forall \sigma.(\sigma F \in \mathcal{C} \Rightarrow \sigma F \preceq F) \ ,$$

where σ ranges over substitutions. That is, every instance of F that meets the syntactic constraint is no larger than F itself.

Any ground (variable-free) formula is trivially mostly-ground. For some definitions of $\preceq$ and some constraints on formulas, however, certain formulas containing variables may also be mostly-ground. Section 5 contains such an example, in which $\mathcal{C}$ limits the possible values for some function arguments to a finite set. Note that, as a consequence of P3$'$, there exist only a finite number of instances (modulo variable renaming) of any mostly-ground formula. The preconditions require that every formula in Γ be mostly-ground, in order to limit the number of new formulas that can be derived through simple instantiation.

Before proceeding with the preconditions, we need to define *unification modulo rewrites* briefly:

Definition 5 *Formulas F_1 and F_2 can be* unified modulo rewrites *if and only if there exists a substitution, σ, of terms for variables such that*

$$W \vdash (\sigma F_1 = \sigma F_2)$$

where W is the set of all rewrites. The substitution σ is called a unifier *for F_1 and F_2.*

With this definition, if F_1 and F_2 can be unified modulo rewrites, then we can prove F_2 from F_1 by applying instances of zero or more rewrites. Except where explicitly noted, unification is always assumed to be modulo rewrites.

Now we define the allowed classes of rules in R: S-rules and G-rules.

Definition 6 *An* S-rule *(shrinking rule) is a rule in which some subset, pri, of the P_i (premises) are designated "primary premises," and for which the conclusion, C, satisfies*

$$\exists P \in pri.(C \preceq P) \ .$$

That is, for an S-rule, the conclusion is no larger than some primary premise. We call the non-primary premises *side conditions*. The premises of an S-rule may be partitioned into primary premises and side conditions in any way such that this condition holds, and the S/G restriction (described later) is also satisfied.

The *G-rules* are applied only as necessary, so it is safe for them to yield formulas larger than their premises. In fact, it is required that the G-rules "grow" in this way, so that backward chaining with them will terminate.

Definition 7 *A* G-rule *(growing rule) is a rule for which*

$$\forall k.(P_k \preceq C)$$

In a G-rule, the conclusion is at least as large as any premise.

The *rewrites* are intended to provide simple transformations that have no effect on the size of a formula. They are often useful for expressing associativity or commutativity of certain functions in the logic. The preconditions require that all rewrites (the set W) be *size-preserving*:

Definition 8 *A rewrite,*

$$\forall \overline{X}.(S = T) \ ,$$

is size-preserving *if $S \preceq T$ and $T \preceq S$.*

From this definition and pre-order conditions P1 and P2, it follows immediately that if we apply a rewrite to a formula, F, producing F', then

$$\forall G.(G \preceq F) \iff (G \preceq F')$$

and

$$\forall G.(F \preceq G) \iff (F' \preceq G)\ .$$

That is, the rewrite has not affected the "size" of F.

Finally, to guarantee termination, the algorithm requires that the *S/G restriction* hold for each S-rule:

Definition 9 S/G restriction: *Given an S-rule with primary premises P_i, side conditions S_i, and conclusion C,*

- *each primary premise P_i must not unify with any G-rule conclusion, and*
- *each side-condition, S_i, must satisfy $S_i \preceq C$.*

Note that this restriction constrains the manner in which S- and G-rules can interact with each other. Whereas the other restrictions are local properties and thus can be checked for each rule, rewrite, and assumption in isolation, this global restriction involves all rules and rewrites. It can, however, be checked quickly and automatically. Along with the S-rule definition, this restriction defines the role of the side conditions: unlike the primary premises, whose instantiations largely determine the form of the S-rule's result, the side conditions serve mainly as qualifications that can prevent or enable a particular application of the S-rule.

We can now define the full precondition that ensures the TG_ℓ algorithm will succeed with a given set of inputs.

Definition 10 *The TG_ℓ precondition holds for some finite sets of assumptions (Γ), rules (R), and rewrites (W); some syntactic constraint ($\mathcal{C}$); and some pre-order ($\preceq$), if and only if all of the following are true:*

- *The pre-order, $\preceq$, satisfies conditions P1, P2, and P3$'$ (given $\mathcal{C}$).*
- *Every formula in Γ is mostly-ground, with respect to $\preceq$.*
- *Every rule in R is either a G-rule or an S-rule, with respect to $\preceq$.*
- *Every rewrite in W is size-preserving, with respect to $\preceq$.*
- *The S/G restriction holds for each S-rule in R.*

Given a pre-order, $\preceq$, that is computable and satisfies the P1-P3$'$ conditions, and a test for mostly-groundness corresponding to $\preceq$, it is possible to check the last four components of this precondition automatically. The partitioning of the rules into S- and G-rules may not be completely determined by the definitions above. If a rule has no premise larger than its conclusion, and the conclusion no larger than any premise, it could go into either category. In some cases, the S/G restriction may determine the choice, but in others it must be made arbitrarily or at the user's suggestion. (A rule whose conclusions are rarely interesting in their own right should probably be designated a G-rule.) The S-rules' primary premises can be identified automatically as those premises that match no G-rule conclusions.

There are ℓ_{RW} logics for which regardless of the $\preceq$ pre-order chosen, the TG_ℓ preconditions cannot hold. Here is one such logic:

$$\mathbf{R1} : \frac{g(f(X))}{g(X)} \qquad \mathbf{R2} : \frac{g(X)}{g(f(X))}$$

(There are no rewrites or syntactic constraints.) To see why this logic can never meet the TG_ℓ preconditions, consider first the case that $\mathbf{R1}$ is an S-rule. Since $\mathbf{R1}$'s premise matches the conclusion of $\mathbf{R2}$, it follows that $\mathbf{R2}$ must also be an S-rule or the S/G restriction would fail. Since $\mathbf{R2}$ is an S-rule, that implies that its conclusion is no larger than its premise:

$$g(f(X)) \preceq g(X)$$

By pre-order condition P2 and reflexivity of pre-orders, all formulas of the form, $g(f(f(\ldots f(X))))$, are $\preceq g(X)$. Since there are infinitely many such formulas, pre-order condition P3 is violated, so we have a contradiction, and $\mathbf{R1}$ must not be an S-rule. The only other possibility is that $\mathbf{R1}$ is a G-rule. From the G-rule definition, though, we have

$$g(f(X)) \preceq g(X)$$

which, again, is impossible, so this pair of rules is unusable with TG_ℓ.

3.2 Algorithm Sketch and a Simple Example

The theory generation algorithm, TG_ℓ, essentially consists of performing *forward chaining* with the S-rules starting from the assumptions, with *backward chaining* at each step to satisfy S-rule premises using G-rules and rewrites. Forward chaining is the repeated application of rules to a set of known formulas, adding new known formulas to this set until a fixed point is reached. Backward chaining is the process of searching for a derivation of some desired formula by applying rules "in reverse," until all premises can be satisfied from known formulas. The basic components of the algorithm—forward chaining, backward chaining, and unification—are widely used methods in theorem proving and planning. We assemble them in a way that takes advantage of the S-rule/G-rule distinction, and that applies rewrites transparently through a modified unification procedure satisfying Definition 5. In the next section, we describe this combined forward/backward chaining approach in detail.

The skeleton of the TG_ℓ algorithm is the exploration of a directed graph which has a node for each formula that will be in the generated theory representation. The roots of this graph are the assumptions, and an edge from F_1 to F_2 indicates that the formula F_1 is used (perhaps via G-rules and rewrites) to satisfy a premise of an S-rule which yields F_2. The algorithm enumerates the nodes in this graph through a breadth-first traversal. At each step of the traversal, we consider all possible applications of the S-rules using the formulas derived so far—the visited nodes of the graph. The new fringe consists of all the new conclusions reached by those S-rule applications. When no new conclusions can be reached, the exploration is complete and the formulas already enumerated constitute the theory representation. We illustrate in Figure 1 the progress from initial assumptions (indicated by the smallest oval), through incremental expansion, converging at a fixed point, the final theory representation (indicated by the darkest oval). We explain the dashed arrows later.

Before describing the algorithm in detail, we present a simple example application of TG_ℓ. The logic we use has one S-rule:

$$\mathbf{S1}: \frac{\mathit{wearing}(P, X) \qquad \mathit{thick}(X)}{\mathit{warm}(P)}$$

one G-rule:

$$\mathbf{G1}: \frac{\mathit{thick}(X)}{\mathit{thick}(\mathit{layered}(X, Y))}$$

and one rewrite:

$$\mathbf{W1}: \mathit{layered}(X, Y) = \mathit{layered}(Y, X)\ .$$

We apply TG_ℓ to these initial assumptions:

$$\mathit{wearing}(\mathit{alice}, \mathit{layered}(\mathit{blouse}, \mathit{sweater}))$$
$$\mathit{thick}(\mathit{sweater})$$

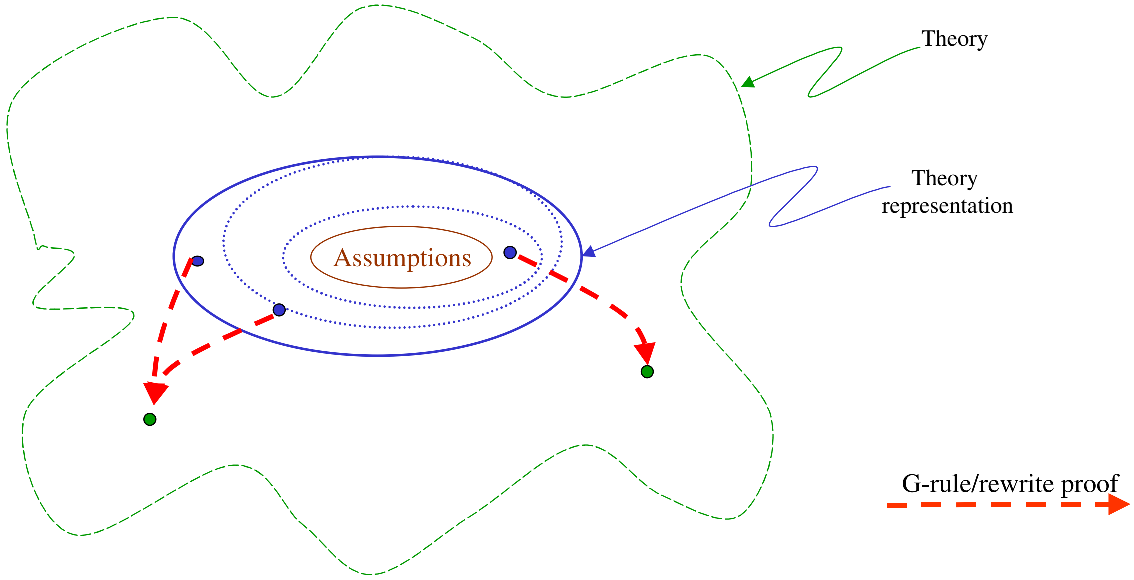

Figure 1: An illustration of the progress of the TG_ℓ algorithm.

First, the algorithm tries to apply rule **S1**; unifying its primary premise with one of the known formulas gives this substitution:

$$P = alice, X = layered(blouse, sweater)$$

To satisfy the other premise, we do backward chaining with the G-rule, starting with

$$thick(layered(blouse, sweater))$$

This matches no known-valid formula directly, so we try reverse-applying the G-rule (**G1**). Unifying the desired formula with **G1**'s conclusion (and using the rewrite, **W1**), we get the two substitutions,

$$X = blouse, Y = sweater$$

and

$$X = sweater, Y = blouse\ .$$

We instantiate **G1**'s premise with the first substitution, and get

$$thick(blouse)$$

which fails to match any known formula or G-rule conclusion. We then instantiate **G1**'s premise with the second substitution, and reach

$$thick(sweater)$$

which matches one of the initial assumptions. Since all **S1**'s premises have been satisfied, we proceed with the application, and we add $warm(alice)$ to the set of known-valid formulas. No further applications of **S1** are possible, so TG_ℓ terminates, producing this theory representation:

$$\begin{array}{l} wearing(alice, layered(blouse, sweater)) \\ thick(sweater) \\ warm(alice) \end{array}$$

Generate the theory representation induced by the given assumptions, S-rules, G-rules, and rewrites, under the pre-order, $\preceq$.
The arguments to $\mathit{theory\_gen}$ are assumed to be in scope in all other functions.

function $\mathit{theory\_gen}(\mathit{Assumptions}, \mathit{S\_rules}, \mathit{G\_rules}, \mathit{Rewrites}, \preceq) =$
  if $\neg \mathit{SG\_restriction\_ok}(\mathit{S\_rules}, \mathit{G\_rules}, \mathit{Rewrites}, \preceq)$ **then**
    raise `BadRules`
  else
    return $\mathit{closure}(\mathit{make\_canonical}(\mathit{Assumptions}), \{\})$

Given a partially-generated theory representation (T) and some new formulas (*fringe*), return the theory representation of $T \cup \mathit{fringe}$.

function $\mathit{closure}(\mathit{fringe}, T) =$
  if $\mathit{fringe} = \{\}$ **then**
    return T
  else
    $T' \leftarrow T \cup \mathit{fringe}$
    $\mathit{fringe}' \leftarrow \bigcup_{R \in \mathit{S\_rules}} \mathit{make\_canonical}(\mathit{apply\_srule}(R, T')) \setminus T'$
    return $\mathit{closure}(\mathit{fringe}', T')$

Apply the given S-rule in every possible way using the formulas in known, with help from the G-rules and rewrites.

function $\mathit{apply\_srule}(R, \mathit{known}) =$
  return $\{\mathit{apply\_subst}(\sigma, \mathit{conclusion}(R))$
    $\mid \sigma \in \mathit{backward\_chain}(\mathit{premises}(R), \mathit{known}, \{\})\}$

Figure 2: Pseudocode description of the TG_ℓ algorithm (part 1 of 2).

Using this theory representation, we can test specific formulas for membership in the theory by simple backward chaining using just $\mathbf{G1}$ and $\mathbf{W1}$. For instance, to test

$$\mathit{thick}(\mathit{layered}(\mathit{sweater}, \mathit{blouse}))$$

we unify this formula with $\mathbf{G1}$'s conclusion and get the new premise,

$$\mathit{thick}(\mathit{sweater})$$

which appears in the theory representation, so the original formula must be in the theory.

3.3 The Algorithm

A pseudocode description of the algorithm appears in Figures 2–3. The $\mathit{theory\_gen}$ function verifies the precondition and invokes $\mathit{closure}$ to generate the theory. The $\mathit{closure}$ function performs the basic breadth-first traversal; it builds up a set of formulas in T which will eventually be the theory representation, while keeping track of a "fringe" of newly added formulas. At each step, $\mathit{closure}$ finds all formulas derivable from $(T \cup \mathit{fringe})$ with a single S-rule application by calling $\mathit{apply\_srule}$ for each S-rule. It then takes the canonical forms of these derivable formulas, and puts any new ones (not already derived) in the new fringe.

The formulas added to T are always canonical representatives of their equivalence classes, under a renaming/rewriting equivalence relation. To be precise, this relation equates formulas F and G if there exists a renaming σ such that the rewrites imply $\sigma F = \sigma G$. (A renaming is a substitution that only substitutes variables for variables.) By selecting these canonical representatives, we prevent some redundancy in the theory representation. The renaming equivalence is actually necessary to ensure termination, since variables can be renamed arbitrarily by the lower-level functions. Requiring that formulas be canonical modulo rewrites, while not strictly necessary for termination, does make the algorithm faster, and perhaps more importantly, makes the theory representation canonical in the sense of criterion C4 (from Section 2.2). The canonical representatives may be chosen efficiently using a simple lexicographical order.

The $apply\_srule$ function simply calls $backward\_chain$ in order to find all ways to satisfy the given S-rule's premises. Note that, as represented here, $closure$ finds all possible S-rule applications and simply ignores the ones that do not produce new formulas. If the $apply\_srule$ function is told which formulas are in the fringe, and can pass this information along to $backward\_chain$, it can avoid many of these redundant S-rule applications, and return only formulas whose derivations make use of some formula from the fringe.

Figure 3 contains the three mutually recursive backward-chaining functions: $backward\_chain$, $backward\_chain\_one$, and $reverse\_apply\_grule$. The purpose of $backward\_chain$ is to satisfy a set of goals in all possible ways, with the help of G-rules and rewrites. It calls $choose\_goal$ to select the first goal to work on (g). Goals which match some G-rule conclusion, and thus may require a deeper search, are postponed until all other goals have been met. (Note that these goals can only arise from G-rule premises or S-rule side-conditions.) The $choose\_goal$ function may apply further heuristics to help narrow the search early.

The $backward\_chain\_one$ function searches for a derivation of a single formula (ϕ) with G-rules and rewrites. It first checks whether a canonical equivalent of ϕ occurs in the $visited$ set, and fails if so, since those formulas are assumed to be unprovable. This occurrence corresponds to a derivation search which has hit a cycle. If ϕ is not in this set, the function renames variables occurring in ϕ uniquely, to avoid conflicts with variables in the G-rules, rewrites, and $known$. It then collects all substitutions that unify ϕ (modulo rewrites) with some formula in $known$, and for each G-rule, calls $reverse\_apply\_grule$ to see whether that G-rule could be used to prove ϕ. Each substitution that satisfies ϕ either directly from $known$ or indirectly via G-rules is returned, composed with the variable-renaming substitution.

In $reverse\_apply\_grule$, we simply find substitutions under which the conclusion of the given G-rule's conclusion matches ϕ, and for each of those substitutions, call $backward\_chain$ recursively to search for derivations of each of the G-rule's (instantiated) premises.

The TG_ℓ algorithm relies on several low-level utility functions, listed in Figure 4. Most of these are simple and require no further discussion, but the $unify$ function is somewhat unusual. Since rewrites can be applied to any subformula, we can most efficiently handle them by taking them into account during unification. We augment a simple unification algorithm by trying rewrites at each recursive step of the unification. In this way, we avoid applying rewrites until and unless they actually have some effect on the unification process. Plotkin described a similar technique for building equational axioms into the unification process [Plo72]. Because of the rewrites, the unification procedure produces not just zero or one, but potentially many "most general unifiers."

The TG_ℓ algorithm described here can be implemented quite straightforwardly, but various optimizations and specialized data structures can be employed to provide greater efficiency if desired. For instance, we can maintain data structures that allow quick identification of all formulas in $known$ that match a given premise, and we can propagate the fringe to $backward\_chain$, as mentioned above, to avoid redundant S-rule applications early. See Kindred's thesis for a discussion of the implementation in REVERE and the set of optimizations it uses [Kin99].

[Find the set of substitutions under which the given goals can be derived from $known$ using G-rules and rewrites, assuming formulas in $visited$ to be unprovable.

function $backward\_chain(goals, known, visited)$ =
    if $goals = \{\}$ **then**
        return $\{[\,]\}$
    else
        $(g, gs) \leftarrow choose\_goal(goals)$
        return $\{compose(\sigma_2, \sigma_1)$
                $\mid \sigma_1 \in backward\_chain\_one(g, known, visited),$
                $\sigma_2 \in backward\_chain(apply\_subst(\sigma_1, gs), known, visited)\}$

[Find the set of substitutions under which ϕ can be derived from $known$ using G-rules and rewrites, assuming formulas in $visited$ to be unprovable.

function $backward\_chain\_one(\phi, known, visited)$ =
    $\hat{\phi} \leftarrow make\_canonical(\phi)$
    if $\hat{\phi} \in visited$ **then**
        return $\{\}$
    else
        $\sigma_r \leftarrow unique\_renaming(\phi)$
        $\phi_r \leftarrow apply\_subst(\sigma_r, \phi)$
        $regular\_substs \leftarrow \bigcup_{F \in known} unify(\phi_r, F)$
        $grule\_substs \leftarrow \bigcup_{R \in G\_rules} reverse\_apply\_grule(R, \phi_r, known, visited \cup \{\hat{\phi}\})$
        return $\{compose(\sigma, \sigma_r) \mid \sigma \in regular\_substs \cup grule\_substs\}$

[Find the set of substitutions under which ϕ can be derived from $known$ by a proof using G-rules and rewrites, and ending with the given G-rule (R), assuming formulas in $visited$ to be unprovable.

function $reverse\_apply\_grule(R, \phi, known, visited)$ =
    return $\{compose(\sigma_4, \sigma_3)$
        $\mid \sigma_3 \in unify(\phi, conclusion(R)),$
        $\sigma_4 \in backward\_chain(apply\_subst(\sigma_3, premises(R)),$
                $known, visited)\}$

Figure 3: Pseudocode description of the TG_ℓ algorithm (part 2 of 2).

3.4 Decision Procedure

Given a generated theory representation for Γ, and a mostly-ground formula, ϕ, the procedure for deciding $\Gamma \overset{?}{\vdash} \phi$ is simply this:

function $derivable(\phi, theory\_rep) =$
    return $backward\_chain(\{\phi\}, theory\_rep, \{\}) \neq \{\}$

This is a simple search for a proof using only G-rules and rewrites, starting with the generated theory representation. As illustrated in Figure 1, any formula in the theory (within the dashed cloud) must be provable

$SG\_restriction\_ok(S, G, R, \preceq)$	Check that the S/G restriction holds for the given rules, rewrites, and pre-order.
$choose\_goal(goals)$	Select a goal to satisfy first, and return that goal and the remaining goals as a pair; prefer goals that match no G-rule conclusions.
$apply\_subst(\sigma, \Phi)$	Replace variables in Φ (a formula or set of formulas), according to the substitution, σ.
$make\_canonical(F)$	Return a canonical representative of the set of formulas equivalent to F modulo rewrites and variable renaming (can also be applied to sets of formulas).
$unify(F, G)$	Return a set containing each most-general substitution, σ, under which the rewrites imply $\sigma F = \sigma G$.
$unique\_renaming(F)$	Return a substitution that replaces each variable occurring in F with a variable that occurs in no other formulas.
$compose(\sigma_1, \sigma_2)$	Return the composition of substitution σ_1 with σ_2.

Figure 4: Auxiliary functions used by the TG_ℓ algorithm.

from the formulas in the theory representation (the dark oval) using only G-rules and rewrites. The dashed arrows represent these proofs for two formulas that lie within the theory but outside the theory representation. The correctness and termination of this decision procedure follow directly from the correctness and termination of the $backward\_chain$ function (Theorems 5 and 6).

This decision procedure satisfies the following property, where $\mathrm{preconds}$ refers to the TG_ℓ preconditions given in Definition 10:

$$\begin{aligned}&\mathrm{preconds}(T^0, S\_rules, G\_rules, Rewrites, \preceq)\\&\qquad\Longrightarrow (\phi \in T^* \Longleftrightarrow derivable(\phi, theory\_gen(T^0)))\end{aligned}$$

That is, for a logic and initial set of formulas, T^0, that satisfy the TG_ℓ preconditions, a mostly-ground formula, ϕ, is in the theory induced by T^0 if and only if the decision procedure returns true, given ϕ and the results of the TG_ℓ algorithm.

This decision procedure is very efficient in practice, since the S-rules can be safely ignored.

4 Analysis of the TG_ℓ Algorithm

In this section, we sketch proofs of correctness and termination for the TG_ℓ algorithm described in Section 3. The full proofs appear in an appendix.

In these arguments, we make use of two restricted forms of proof within ℓ_{RW}. We write,

$$\Gamma \vdash_{\mathrm{W}} \phi$$

if there exists a proof of ϕ from Γ using only rewrites (and instantiation). If there exists a proof of ϕ from Γ using rewrites and G-rules, we write,

$$\Gamma \vdash_{\mathrm{GW}} \phi\ .$$

4.1 Correctness

The proof of correctness for TG_ℓ has two parts: soundness and completeness. The soundness of TG_ℓ implies that $closure$ only produces formulas that are in the appropriate theory representation; completeness implies that every formula in the theory representation is returned by $closure$. The proofs are largely done by induction on either the number of recursive calls to a function, or the total size of a set of proofs.

Theorem 1 (*backward_chain* soundness) *Let Φ and V be sets of formulas of ℓ_{RW}, let Γ be a set of formulas of ℓ_{RW}, and let σ be a substitution, such that*

$$\sigma \in backward\_chain(\Phi, \Gamma, V) \; .$$

Then, for every $\phi \in \Phi$,

$$\Gamma \vdash_{GW} \sigma\phi \; .$$

This theorem can be proved by induction on the recursion depth. Referring to Figures 2–3, we follow the call to $backward\_chain\_one$, and find two cases. First, the substitution, σ_1, may come from $regular\_substs$, in which case the assumed soundness of $unify$ directly applies. Second, σ_1 may come from $grule\_substs$, in which case we follow the calls through $reverse\_apply\_grule$ and back to $backward\_chain$, so we can apply the induction hypothesis. After a bit of substitution manipulation, the proof is done.

Theorem 2 (*closure* soundness) *Let Γ and Γ' be sets of formulas of ℓ_{RW}. For any formula, F, where*

$$F \in closure(\Gamma', \Gamma) \; ,$$

there exists a proof, $\mathcal{P}$, of $\Gamma \cup \Gamma' \vdash F$, in which the last rule application (if any) is of an S-rule.

The formulas returned by $closure$ are just rewrite-canonicalized versions of those returned by $apply\_srule$. Using Theorem 1, we can show that for each substitution returned by $backward\_chain$, the instantiated premises of the S-rule, R (see Figure 2), can be proved from $\Gamma \cup \Gamma'$, and thus the instantiated conclusion of R has a proof in which the last rule applied is the S-rule, R.

This concludes the soundness side of the correctness proof; the following completeness arguments show that for any proof in ℓ_{RW} whose last rule application uses an S-rule, the conclusion (or a rewrite-equivalent thereof) will appear in the theory representation generated by TG_ℓ.

Theorem 3 (*backward_chain* completeness) *Let Γ and V be sets of formulas, let Φ be a set of formulas ($\{\phi_1, \ldots, \phi_n\}$), let σ be a substitution, and let $\mathcal{P}_1 \ldots \mathcal{P}_n$ be proofs (using no S-rules), such that for $1 \le i \le n$,*

$$\Gamma \overset{\mathcal{P}_i}{\vdash}_{GW} \sigma\phi_i$$

and the proofs, $\mathcal{P}_i$, contain no rewrites of formulas in V. Then, if $backward\_chain$ terminates, there exists a substitution, σ', such that

$$\sigma' \in backward\_chain(\Phi, \Gamma, V)$$

and σ is an extension of σ'.

This theorem is proved by induction on the total number of G-rule applications in the proofs, $\mathcal{P}_i$. This corresponds to the depth of the function's recursion. We show how each G-rule application in the proof must have a corresponding successful application of $reverse\_apply\_grule$, and that all the substitutions returned compose properly. The only wrinkle is that we must ensure that the use of the visited set, V, does not exclude any needed substitutions. Essentially, in any case where the algorithm short-circuits by finding its current goal in the visited set, we can show that a shorter, equivalent proof must exist.

In stating the next theorem, we use a notion of *partial theory representation*, which is simply a $\langle \mathit{fringe}, T \rangle$ pair which represents a valid intermediate step in application of the $\mathit{closure}$ function, satisfying the obvious invariant.

Theorem 4 ($\boldsymbol{closure}$ completeness) *Let T and fringe be sets of formulas (of ℓ_{RW}), such that $\langle \mathit{fringe}, T \rangle$ is a partial theory representation. For any formula, ϕ, and proof, $\mathcal{P}$, whose last rule application is an S-rule application, where*

$$(T \cup \mathit{fringe}) \overset{\mathcal{P}}{\vdash} \phi \ ,$$

there exists some ϕ', where (assuming $\mathit{closure}$ terminates)

$$\phi' \in \mathit{closure}(\mathit{fringe}, T) \ ,$$

such that

$$\phi' \vdash_W \phi \ .$$

This theorem is proved by induction on the recursion depth. Using Theorem 3, we can show that $\mathit{apply\_srule}$ will satisfy the premises of the S-rule, R, in all ways possible using G-rules and rewrites. With the induction hypothesis, this ensures that for any proof ending with an S-rule application, that proof's rewrite-canonicalized conclusion will be added to the theory representation.

Finally, we can show that TG_ℓ produces exactly the desired theory representation, if it terminates.

Theorem 5 (TG_ℓ **Correctness)** *If Γ is a set of mostly-ground formulas of ℓ_{RW}, and S_rules, G_rules, $Rewrites$, and $\preceq$ meet the TG_ℓ algorithm preconditions given in Definition 10, then*

$$\mathit{theory\_gen}(\Gamma, S\_rules, G\_rules, Rewrites, \preceq)$$

returns an (R, R') representation of the theory induced by Γ, where R' is the set of S-rules, and the equivalence used is equivalence modulo rewrites and variable renaming.

This follows simply from Theorems 2 and 4.

4.2 Termination

The completeness proofs above hold only when the TG_ℓ algorithm terminates, so it remains to show that it always does. The proof goes, roughly, as follows.

The $\mathit{backward\_chain}$ function first satisfies the primary premises, and then applies G-rules in reverse to satisfy the partially instantiated side-conditions. Since the G-rules "grow" when applied in the forward direction, they "shrink" when applied in reverse (if the goal formula is sufficiently ground), so the size of the subgoals produced through this backward chaining can be bounded, given the goals and known-valid formulas it starts with. Furthermore, since it maintains a *visited* set and checks goals against that set, the recursion depth is limited by the number of unique formulas within that bound. We know this number is finite, from the pre-order conditions.

The $\mathit{closure}$ function finds each way of applying the S-rules with help from $\mathit{backward\_chain}$, and repeats until it reaches a fixed point. Since the S-rules "shrink," they can never produce a formula larger than all the initial assumptions, and so this process will halt as well.

The termination theorem itself is simply the following:

Theorem 6 *If the TG_ℓ preconditions hold, then $\mathit{theory\_gen}$ will terminate.*

The proof makes use of the notion of *size-boundedness*:

Definition 11 *A formula, F, is* size-bounded *by a finite set of formulas, Γ, when, for any substitution, σ, there exists $G \in \Gamma$ such that*

$$\sigma F \preceq G \ .$$

Note that if F is mostly-ground then F is size-bounded by $\{F\}$.

Since the initial assumptions given to $theory\_gen$ are mostly-ground, they are size-bounded by themselves. We can show that at each function invocation during the course of the algorithm, this size bound is preserved. Let the initial assumptions be size-bounded by Γ. The formulas passed to $closure$ are rewrite-canonicalized versions of these assumptions; rewrites are size-preserving, so the formulas in $fringe$ (and trivially, T) are still size-bounded by Γ.

At each iteration of $closure$, the size-bound on $fringe$ and T is clearly preserved if $apply\_srule$ preserves the size-bound.

To show that $apply\_srule$ will preserve the size-bound on $known$, we establish a chain of inequalities (using the pre-order, $\preceq$). From the S-rule definition and pre-order condition P2, the S-rule, R, must have some primary premise, P_i, such that

$$\sigma C \preceq \sigma P_i \ ,$$

for any σ. Since in order for $backward\_chain$ to return any substitutions, the P_i premise must unify with some formula, F, in $known$, it follows that there exists a σ (unifying F with P_i) such that

$$\sigma P_i \preceq \sigma F \ ,$$

and furthermore that the substitutions returned by $backward\_chain$ will be extensions of some such σ. From the definition of size-boundedness, it follows from these two relations that for any σ returned by $backward\_chain$,

$$\sigma C \preceq \sigma F \ ,$$

so the size-bound is preserved. Since $closure$ does variable-name canonicalization, it is straightforward to show that it will terminate given the preservation of the size-bound, since pre-order condition P3$'$ ensures that there are finitely many formulas that are size-bounded by Γ and canonical with respect to variable naming.

By a similar argument, the backward-chaining process itself must terminate since the goals used at each step of the recursion are size-bounded by Γ as well, and so the recursion must terminate since variable-name canonicalization and the $visited$ set are used.

The result of this theorem, together with the TG_ℓ correctness theorem, guarantees that, when its preconditions are satisfied, the TG_ℓ algorithm will always produce the desired theory representation.

5 Application to Belief Logics

"Little logics" have been used successfully to describe, analyze, and find flaws in cryptographic protocols. Burrows, Abadi, and Needham developed their (little) logic of authentication around the notion of belief. Derivatives of their BAN logic include GNY [GNY90], SVO [SvO94], AUTLOG [KW94], and Kailar's logic of accountability [Kai96]. In these belief logics, each message in a protocol is represented by a set of beliefs it is meant to convey, and principals acquire new beliefs when they receive messages, according to a small set of rules. The BAN logic allows reasoning not just about the authenticity of a message (the identity of its sender), but also about *freshness*, a quality attributed to messages that are believed to have been sent recently. This allowed BAN reasoning to uncover certain replay attacks like the well-known flaw in the Needham-Schroeder shared-key protocol [DS81].

There has been little in the way of tools for automated reasoning with these logics, however. Some of the BAN analyses were mechanically verified, and the designers of AUTLOG produced a prover for their logic, but prominent automated tools, such as the NRL Protocol Analyzer and Paulson's Isabelle work, have used very different approaches. The lack of emphasis on automation for these logics results in part from their apparent simplicity; it can be argued that proofs are easily carried out by hand. Indeed, the proofs rarely require significant ingenuity once appropriate premises have been established. Manual proofs, however, even in published work often miss significant details and assume preconditions or rules of inference that are not made explicit; automated verification keeps us honest. Furthermore, with fast, automated reasoning we can perform some analyses that would otherwise be impractical or cumbersome, such as enumerating beliefs held as the protocol progresses.

The development of theory generation was partially motivated by the need for automated reasoning with this family of logics. Using theory generation, and the TG_ℓ algorithm in particular, we can do automated reasoning for all these logics with a single, simple tool: the REVERE system. REVERE is a protocol analysis tool built around a theory generation core. It has plug-in modules expressing different logics, and accepts protocol specifications written in a variant of the Common Authentication Protocol Specification Language (CAPSL) [Mil97].

We examine in Section 5.1 the BAN logic of authentication, and in Section 5.2.1, three other logics in the BAN family: AUTLOG, Kailar's logic of accountability, and our new belief logic called RV. We show in our use of REVERE how theory generation can be applied to each of them.

5.1 The BAN Logic

As the progenitor of this family, the BAN logic of authentication is a natural case to consider first. This logic is normally applied to authentication protocols. It allows certain notions of belief and trust to be expressed in a simple manner, and it provides rules for interpreting encrypted messages exchanged among the parties (principals) involved in a protocol. In Section 5.1.1 we enumerate the fundamental concepts expressible in the BAN logic: belief, trust, message freshness, and message receipt and transmission; Section 5.1.2 describes the rules of inference; Sections 5.1.3–5.1.4 give examples of their application.

5.1.1 Components of the Logic

In encoding the BAN logic and its accompanying sample protocols, we must make several adjustments and additions to the logic as originally presented [BAN90], to account for rules, assumptions, and relationships that are missing or implicit.

Figure 5 shows the functions used in the encoding, and their intuitive meanings. The first twelve correspond directly to constructs in the original logic, and have clear interpretations. The last two are new: inv makes explicit the relationship implied between the keys in a key pair (K, K^{-1}) under public-key (asymmetric) cryptography, and $distinct$ expresses that two principals are not the same.

As a technical convenience, we always assume that for every function (e.g., $believes$), there is a corresponding predicate by the same name and with the same arity, which is used when the operator occurs at the outermost level. For instance, in the BAN formula

$$A \textbf{ believes } B \textbf{ said } A \textbf{ believes } A \xleftrightarrow{K} B$$

the first **believes** is represented by the $believes$ predicate, while the second is represented by the $believes$ function.

Function	BAN notation	Meaning
$believes(P, X)$	P **believes** X	P believes statement X
$sees(P, X)$	P **sees** X	P sees message X
$said(P, X)$	P **said** X	P said message X
$controls(P, X)$	P **controls** X	if P claims X, X can be believed
$fresh(X)$	**fresh**(X)	X has not been uttered before this protocol run
$shared\_key(K, P, Q)$	$P \overset{K}{\leftrightarrow} Q$	K is a symmetric key shared by P and Q
$public\_key(K, P)$	$\overset{K}{\mapsto} P$	K is P's public key
$secret(Y, P, Q)$	$P \overset{Y}{\rightleftharpoons} Q$	Y is a secret shared by P and Q
$encrypt(X, K, P)$	$\{X\}_K$ *from* P	message X, encrypted under key K by P
$combine(X, Y)$	$\langle X \rangle_Y$	message X combined with secret Y
$comma(X, Y)$	X, Y	concatenation
$A, B, S, T, \ldots$	$A, B, S, T, \ldots$	0-ary functions (constants)
$inv(K_1, K_2)$		K_1 and K_2 are a public/private key pair
$distinct(P, Q)$		principals P and Q are not the same

Figure 5: BAN functions

The above formula should be parsed to read as "A believes that (B said that (A believes that (A and B share the key K)))." Typically A would come to having this belief by receiving from B a message whose contents conveys the belief that A and B share K.

We provide a finite but unspecified set of uninterpreted 0-ary functions (constants), which can be used to represent principals, keys, timestamps, and so forth in a specific protocol description.

The pre-order we use for the BAN logic (to ensure termination of TG_ℓ) is a relatively simple one, in which $F \preceq G$ when F contains no more symbols than G, and the number of occurrences of each variable in F is no larger than the number in G. There is one exception to this: various functions have *atomic arguments* in which variables and symbols are not counted; these arguments correspond to principal names. A syntactic constraint ensures that in all formulas, these atomic arguments will always contain either a single variable or a constant (0-ary function). The formal pre-order definition is the following:

Definition 12 *The pre-order, $\preceq_{BAN}$, is defined over BAN terms and formulas as follows:*

$$\begin{array}{rcl} F \preceq_{BAN} G & \equiv & (nsyms(F) \leq nsyms(G)) \\ & & \wedge\, (\forall v. occ(v, F) \leq occ(v, G)) \\ nsyms(F) & \equiv & \textit{the number of functions, predicates, and variables in } F\textit{,} \\ & & \textit{excluding those in atomic arguments} \\ occ(v, F) & \equiv & \textit{the number of occurrences of variable } v \textit{ in } F\textit{, excluding} \\ & & \textit{occurrences in atomic arguments} \end{array}$$

This pre-order satisfies conditions P1–P3$'$ (from Section 3.1) [Kin99].

5.1.2 Rules of Inference

The BAN logic contains eleven basic rules of inference, each of which can be expressed as an ℓ_{RW} rule, written in the form

$$\frac{P_1, P_2, \ldots, P_m}{C}$$

Of these rules, ten are an S-rules under the BAN pre-order, the eleventh is a G-rule, and each preserves the atomic-arguments constraint on BAN formulas.

In order to make the published BAN analysis of the Andrew Secure RPC protocol go through, we had to add an extra S-rule which allows deriving both the freshness and the authenticity of a message whose freshness is provided by the freshness of the encryption key:

$$\frac{\begin{array}{c} believes(P, fresh(K)) \\ sees(P, encrypt(X, K, R)) \\ distinct(P, R) \\ believes(P, shared\_key(K, Q, P)) \end{array}}{believes(P, believes(Q, X))}$$

This rule does the work of BAN's "message-meaning" and "nonce-verification" rules simultaneously. The normal BAN message-meaning rule throws away the information necessary to prove freshness after demonstrating authenticity. (For completeness, we also added a similar rule dealing with authentication via secrets.)

In addition, we added seven "freshness" G-rules, which allow deriving, for example, the freshness of an encrypted message from the freshness of the encryption key. While not strictly necessary, these rules allow more straightforward freshness assumptions in protocol specifications. Finally, we added two rules for breaking conjunctions that are mentioned indirectly in the BAN paper, and both of which appear in a technical report by the same authors [BAN89].

Having encoded the rules, we can analyze each of the four protocols examined in the BAN paper and check all the properties claimed there [BAN90].

5.1.3 Kerberos Example

Through a sequence of four messages, the Kerberos protocol establishes a shared key for communication between two principals, using a trusted server [MNSS87]. The simplified concrete protocol assumed in the original BAN analysis is the following:

Message 1. $A \rightarrow S : A, B$
Message 2. $S \rightarrow A : \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
Message 3. $A \rightarrow B : \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$
Message 4. $B \rightarrow A : \{T_a + 1\}_{K_{ab}}$

Initially, A wants to establish a session key for secure communication with B. A sends Message 1 to the trusted server, S, as a hint that she wants a new key to be shared with B. The server responds with Message 2, which is encrypted with K_{as}, a key shared by A and S. In this message, S provides the new shared key, K_{ab}, along with a timestamp (T_s), the key's lifetime (L), B's name, and an encrypted message intended for B. In Message 3, A forwards this encrypted message along to B, who decrypts the message to find K_{ab} and its associated information. In addition, A sends a timestamp (T_a) and A's name, encrypted under the new session key, to demonstrate to B that A has the key. Finally, B responds with Message 4, which is simply $T_a + 1$ encrypted under the session key, to show A that B has the key as well.

The BAN analysis of this protocol starts by constructing a three-message *idealized protocol* [BAN90]; the idealized protocol ignores Message 1, since it is unencrypted and thus cannot safely convey any beliefs. The BAN analysis then goes on to list ten initial assumptions regarding client/server shared keys, trust of the server, and freshness of the timestamps used [BAN90]. We express each of these three messages and ten assumptions directly (the conversion is purely syntactic), and add four more assumptions (see Figures 6 and 7).

Message 2. $S \to A : \{T_s, A \stackrel{K_{ab}}{\longleftrightarrow} B, \{T_s, A \stackrel{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$

$$\begin{aligned} &\mathit{sees}(A, \mathit{encrypt}(\mathit{comma}(\mathit{comma}(T_s, \mathit{shared\_key}(K_{ab}, A, B)),\\ &\qquad\qquad \mathit{encrypt}(\mathit{comma}(T_s, \mathit{shared\_key}(K_{ab}, A, B)),\\ &\qquad\qquad\qquad K_{bs}, S))),\\ &\quad K_{as}, S)) \end{aligned}$$

Message 3. $A \to B : \{T_s, A \stackrel{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}, \{T_a, A \stackrel{K_{ab}}{\longleftrightarrow} B\}_{K_{ab}}$ *from* A

$$\begin{aligned} &\mathit{sees}(B, \mathit{comma}(\mathit{encrypt}(\mathit{comma}(T_s, \mathit{shared\_key}(K_{ab}, A, B)), K_{bs}, S),\\ &\qquad \mathit{encrypt}(\mathit{comma}(T_a, \mathit{shared\_key}(K_{ab}, A, B)), K_{ab}, A))) \end{aligned}$$

Message 4. $B \to A : \{T_a, A \stackrel{K_{ab}}{\longleftrightarrow} B\}_{K_{ab}}$ *from* B

$$\mathit{sees}(A, \mathit{encrypt}(\mathit{comma}(T_a, \mathit{shared\_key}(K_{ab}, A, B)), K_{ab}, B))$$

Figure 6: Kerberos protocol messages, in BAN idealized form and converted to the syntax of our encoding.

$$\begin{aligned} &\mathit{believes}(A, \mathit{shared\_key}(K_{as}, S, A))\\ &\mathit{believes}(B, \mathit{shared\_key}(K_{bs}, S, B))\\ &\mathit{believes}(S, \mathit{shared\_key}(K_{as}, A, S))\\ &\mathit{believes}(S, \mathit{shared\_key}(K_{bs}, B, S))\\ &\mathit{believes}(S, \mathit{shared\_key}(K_{ab}, A, B))\\ &\mathit{believes}(A, \mathit{controls}(S, \mathit{shared\_key}(K_{ab}, A, B)))\\ &\mathit{believes}(B, \mathit{controls}(S, \mathit{shared\_key}(K_{ab}, A, B)))\\ &\mathit{believes}(A, \mathit{fresh}(T_s))\\ &\mathit{believes}(B, \mathit{fresh}(T_s))\\ &\mathit{believes}(B, \mathit{fresh}(T_a)) \end{aligned}$$

$$\begin{aligned} &\mathit{believes}(A, \mathit{fresh}(T_a))\\ &\mathit{distinct}(A, S)\\ &\mathit{distinct}(A, B)\\ &\mathit{distinct}(B, S) \end{aligned}$$

Figure 7: Encoding of the Kerberos initial assumptions. All but the last four assumptions appear in the BAN analysis [BAN90].

The first extra assumption—that A must believe its own timestamp to be fresh—is missing in the original paper, and the last three are required to satisfy the distinctness side-conditions. After making these adjustments, we can run the 14 initial assumptions and 3 messages through the TG_ℓ algorithm, and it produces an additional 50 true formulas.

By running the simple decision procedure described in Section 3, we can verify that these four desired properties hold:

$$believes(A, shared\_key(K_{ab}, A, B))$$
$$believes(B, shared\_key(K_{ab}, A, B))$$
$$believes(B, believes(A, shared\_key(K_{ab}, A, B)))$$
$$believes(A, believes(B, shared\_key(K_{ab}, A, B)))$$

These results agree with the original BAN analysis. The first two indicate that each of the two parties believes it shares a key with the other, and the second two, that each believes that the other believes the same thing.

If we remove the optional final message from the protocol and run the algorithm again, it generates 41 valid formulas. By computing the difference between this set and the first set of 50, we can determine exactly what the final message contributes. Among the 9 formulas in this difference is

$$believes(A, believes(B, shared\_key(K_{ab}, A, B)))$$

(the last of the four results above). This confirms the claim in the original analysis that "the three-message protocol does not convince A of B's existence" [BAN90]. This technique of examining the set difference between the deduced properties of two versions of a protocol is a simple but powerful benefit of the theory generation approach; it helps in understanding differences between protocol variants and it supports rapid prototyping during protocol design.

5.1.4 Other Authentication Protocols Using BAN

We encoded the assumptions and messages of the three variants of the Andrew secure RPC handshake protocol given in the BAN paper, and the TG_ℓ algorithm produces the expected results. The last of these verifications requires an extra freshness assumption mentioned indirectly in the BAN analysis:

$$believes(B, fresh(K'_{ab}))$$

It also makes use of one of our added freshness rules and the first simultaneous message-meaning/nonce-verification rule.

We ran the algorithm on two variants of the CCITT X.509 protocol explored in the BAN paper. One of these checks failed to produce the expected results, and this led to the discovery of an oversight in the BAN analysis: they observe a weakness in the original X.509 protocol and claim, "The simplest fix is to sign the secret data Y_a and Y_b before it is encrypted for privacy." In fact we must sign the secret data together with a nonce to ensure freshness. We replace the occurrence of Y_a in the original protocol by

$$encrypt(comma(Y_a, T_a), K'_a, A)$$

and the occurrence of Y_b by

$$encrypt(comma(Y_b, N_a), K'_b, B) \; .$$

After correcting this, the verifications proceed as expected.

Finally, we also duplicated the BAN results for two variants of the Needham-Schroeder public-key secret-exchange protocol, the Wide-Mouth Frog protocol, and the Yahalom protocol.

5.2 Other Logics

5.2.1 AUTLOG

AUTLOG is an extension of the BAN logic, proposed by Kessler and Wedel [KW94]. It incorporates several new concepts, some of which appear in other BAN variants, such as the GNY logic developed by Gong, Needham, and Yahalom [GNY90]. It allows analysis of a simulated eavesdropper for detecting some information leaks, uses the notion of principals "recognizing" decrypted messages, and introduces a "recently said" notion which is more precise than BAN's beliefs about beliefs.

Our encoding of AUTLOG uses all the BAN functions, and a few extras, listed in Figure 8. The original

Function
$recognizable(X)$
$mac(K, X)$
$hash(X)$
$recently\_said(P, X)$

Figure 8: Extra AUTLOG functions

rules of inference from AUTLOG can be entered almost verbatim. There are 22 S-rules and 24 G-rules; the rules governing freshness and recognition are the only G-rules. In applying the TG_ℓ algorithm, we can use a similar pre-order for AUTLOG to that used for the BAN logic. In encoding the AUTLOG rules and attempting some REVERE protocol analyses using them, we found three flaws in the logic, corresponding to six rules that were unsound and one that was weaker than necessary.

One flaw is in AUTLOG's four "key" rules (K1–K4). These rules allow a principal, P to determine, for example, that another principal, Q, considers a key shared with P to be valid if Q has used the key recently. The "recency" is provided by the premise,

$$believes(P, recently\_said(Q, X)) ,$$

where X is the body of the encrypted message. However, that premise is too weak to ensure that the encrypted message was actually sent recently; a more appropriate premise would be

$$believes(P, fresh(X)) .$$

AUTLOG can represent message authentication codes (MACs, or keyed hashes), but the rule that authenticates these codes (A2) is weaker than it should be. It includes the premise that the recipient, P, has seen the message from which the MAC was taken. As we found when our initial SKID2 protocol verification failed, this fails to allow for a situation in which the message has two parts, each of which P has seen. To fix this, we added a new operator and five G-rules for determining what messages a principal is capable of constructing, and replaced the too-strong premise.

The third flaw appears in two of the "recognizing" rules. A principal is said to recognize some message roughly when it can distinguish that message from random data. The recognizing rules for cryptographic hash functions and message authentication codes (MACs), however, state that a principal recognizes a hash if it recognizes the original message from which the hash was built. If that principal has never seen all parts of the original message, though, it will be unable to compute the hash, and thus cannot recognize it. These rules can be fixed by adding a premise requiring that the principal can construct the original message.

To check a protocol for leaks using AUTLOG, one finds the consequence closure over the "seeing" rules of the transmitted messages. The resulting list will include everything an eavesdropper could see. The TG_ℓ

algorithm is well-suited to computing this list; the seeing rules are all S-rules, so the algorithm will generate exactly the desired list.

Kessler and Wedel present two simple challenge-response protocols: one in which only the challenge is encrypted and another in which only the response is encrypted. We have encoded both of these protocols and verified the properties Kessler and Wedel claim: that both achieve the authentication goal

$$\mathit{believes}(B, \mathit{recently\_said}(A, R_B))$$

where R_B is the secret A provides to prove its identity. Furthermore, through the eavesdropper analysis mentioned above, we can show that in the encrypted-challenge version, the secret is revealed and thus the protocol is insecure. (The BAN logic cannot express this.)

We have also checked that the Kerberos protocol and SKID2/SKID3 authentication protocols [BP95], expressed in AUTLOG, satisfy properties similar to those described in Section 5.1.3.

5.2.2 Kailar's Accountability Logic

More recently, Kailar has proposed a simple logic for reasoning about accountability in electronic commerce protocols [Kai96]. The central construct in this logic is

$$P \ \mathbf{CanProve} \ X$$

which means that principal P can convince anyone in an intended audience sharing a set of assumptions, that X holds, without revealing any "secrets" other than X itself.

Kailar provides different versions of this logic, for "strong" and "weak" proof, and for "global" and "nonglobal" trust. These parameters determine what evidence will constitute an acceptable proof of some claim. The logic we choose uses strong proof and global trust, but the other versions would be equally easy to encode. The encoding uses these functions: $\mathit{CanProve}$, $\mathit{IsTrustedOn}$, $\mathit{Implies}$, $\mathit{Authenticates}$, Says, $\mathit{Receives}$, $\mathit{SignedWith}$, comma, and inv.

We encode the four main rules of the logic as follows:

$$\mathbf{Conj}: \frac{\mathit{CanProve}(P, X),\ \mathit{CanProve}(P, Y)}{\mathit{CanProve}(P, \mathit{comma}(X, Y))}$$

$$\mathbf{Inf}: \frac{\mathit{CanProve}(P, X),\ \mathit{Implies}(X, Y)}{\mathit{CanProve}(P, Y)}$$

$$\mathbf{Sign}: \frac{\begin{array}{c}\mathit{Receives}(P, \mathit{SignedWith}(M, K^{-1}))\\ \mathit{CanProve}(P, \mathit{Authenticates}(K, Q))\\ \mathit{Inv}(K, K^{-1})\end{array}}{\mathit{CanProve}(P, \mathit{Says}(Q, M))}$$

$$\mathbf{Trust}: \frac{\begin{array}{c}\mathit{CanProve}(P, \mathit{Says}(Q, X))\\ \mathit{CanProve}(P, \mathit{IsTrustedOn}(Q, X))\end{array}}{\mathit{CanProve}(P, X)}$$

The **Conj** and **Inf** rules allow building conjunctions and using initially-assumed implications. The **Sign** and **Trust** rules correspond roughly to the BAN logic's public-key message-meaning and jurisdiction rules. We can again use a pre-order similar to that used for BAN. This makes **Conj** a G-rule; the other three are S-rules. There are a total of six S-rules, one G-rule, and three rewrites in our encoding of this logic; the extra S-rules and rewrites do simple comma-manipulation.

IBS protocol messages:

Message 3. $E \to S : \{\{Price\}_{K_s^{-1}}, Price\}_{K_e^{-1}}$

$$Receives(S, SignedWith(comma(SignedWith(Price, K_s^{-1}), Price), K_e^{-1}))$$

Message 5. $S \to E : \{Service\}_{K_s^{-1}}$

$$Receives(E, SignedWith(Service, K_s^{-1}))$$

Message 6. $E \to S : \{ServiceAck\}_{K_e^{-1}}$

$$Receives(S, SignedWith(ServiceAck, K_e^{-1}))$$

Initial assumptions:

$$CanProve(S, Authenticates(K_e, E))$$
$$Implies(Says(E, Price), AgreesToPrice(E, pr))$$
$$Implies(Says(E, ServiceAck), ReceivedOneServiceItem(E))$$

Figure 9: Excerpt from IBS protocol and initial assumptions.

We can replace the construct

$$X \ in \ M$$

(representing interpretation of part of a message) with three explicit rules for extracting components of a message. We add rewrites expressing the commutativity and associativity of $comma$, as in the other logics.

We have verified the variants of the IBS (NetBill) electronic payment protocol that Kailar analyzes. Figure 9 contains an encoding of part of the "service provision" phase of the asymmetric-key version of this protocol. The customer, E, first sends the merchant, S, a message containing a price quote, signed by the merchant; this message is itself signed by the customer to indicate his acceptance of the quoted price. The merchant responds by providing the service itself (some piece of data), signed with her private key. The last message of this phase is an acknowledgement by the customer that he received the service, signed with the customer's private key.

When we run the TG_ℓ algorithm on these messages and assumptions, it applies the **Sign** rule to produce these two formulas:

$$CanProve(S, Says(E, comma(SignedWith(Price, K_s^{-1}), Price)))$$
$$CanProve(S, Says(E, ServiceAck))$$

These conclusions are not particularly noteworthy in their own right; they reflect the fact that S (the seller) can prove that E (the customer) has presented two specific messages. With these formulas, the TG_ℓ algorithm next applies a comma-extracting rule to produce

$$CanProve(S, Says(E, SignedWith(Price, K_s^{-1})))$$
$$CanProve(S, Says(E, Price))$$

This shows that S can prove E sent individual components of the earlier messages. Finally, TG_ℓ applies **Inf** to derive these results, which agree with Kailar's [Kai96]:

$$\mathit{CanProve}(S, \mathit{Says}(E, \mathit{ReceivedOneServiceItem}(E)))$$
$$\mathit{CanProve}(S, \mathit{Says}(E, \mathit{AgreesToPrice}(E, pr)))$$

These represent two desired goals of the protocol: that the seller can prove the customer received the service, and that the seller can prove what price the customer agreed to. The TG_ℓ algorithm stops at this point, since no further rule applications can produce new formulas.

We have verified the rest of Kailar's results for two variants of the IBS protocol and for the SPX Authentication Exchange protocol.

5.2.3 RV Logic

Beyond these existing logics, we have developed a new logic, RV [Kin99], which allows more grounded reasoning about protocols, in that the mapping from concrete messages to abstract meanings is made explicit. Using theory generation, we have applied this logic to several existing protocols, checking honesty, secrecy, feasibility, and interpretation validity properties. These properties are not fully addressed by other belief logics, and they are critical in that failure to check them can lead (and has led) to vulnerabilities. By applying TG_ℓ to the RV logic, we can reveal the Needham-Schroeder public key protocol flaw that was discovered by Lowe [Low96], which traditional BAN analysis did not (and cannot) expose. We can also check that an optimized version of the Woo-Lam protocol [WL92a, WL92b] with fewer messages and less encryption than the original version preserves honesty, secrecy, and belief properties of the original protocol; BAN analysis would be insufficient to demonstrate the safety of the improved protocol.

Like other belief logics, RV takes a constructive approach to protocol verification, in that it focuses on deriving positive protocol properties, rather than searching for attacks. Through extending this approach to honesty and secrecy properties, RV can expose flaws that correspond to concurrent-run attacks without explicitly modeling either an intruder or some set of runs.

5.3 Summary of Performance Results

The table in Figure 10 contains a summary of the results mentioned in this section. Each line in the table shows, for a given protocol, the number of initial assumptions and messages fed to the TG_ℓ algorithm, the number of formulas in the theory representation it generated, and the elapsed time in seconds for generating the representations. In each case, we were able to use theory generation to prove that the protocols satisfied (or failed to satisfy) various desired belief properties. Note that the generated theory representations typically contain on the order of several dozens of formulas. All timings were done on an Digital AlphaStation 500, with 500MHz Alpha 21164 CPU.

6 Related Work

There is a rich history of research on the use of mechanized verification tools—namely theorem provers and model checkers—for reasoning about security. We survey here the most relevant work, focusing on the important differences between existing approaches and theory generation as applied to security protocols.

6.1 Theorem Proving

General-purpose automated theorem proving is the more traditional approach to verifying security properties. Early work on automated reasoning about security made use of the Affirm [GMT+80], HDM [LRS79],

Logic	Protocol	Assumps.	Msgs.	Th. Rep.	TG Time (s)
BAN	Kerberos	14, 13	3	61, 52	4.7
	Andrew RPC	8, 8, 7	4	32, 39, 24	3.2
	Needham-Schroeder	19, 19	5	41, 41	1.5
	CCITT X.509	13, 12	3	69, 74	23.8
	Wide-Mouth Frog	12, 12	2	34, 34	19.3
	Yahalom	9, 17, 17	5	40, 60, 62	23.0
AUTLOG	challenge-response 1	2	2	10	0.3
	challenge-response 2	4	2	13	0.3
	Kerberos	18	3	79	11.3
	SKID3	12	3	41	3.7
Kailar's Accountability	IBS variant 1	14	7	44, 39	0.3
	IBS variant 2	20	7	46, 52	0.3
	SPX Auth. Exchange	18	3	36	0.2
RV	Needham-Schroeder (Pub)	31	7	83	23.0
	Otway-Rees	28	4	95	34.4
	Denning-Sacco	25	3	77	38.1
	Neuman-Stubblebine	35	4	80	20.1
	Woo-Lam	37	7	106	50.8

Figure 10: Protocol analyses performed with existing belief logics, with the number of formulas in the initial assumptions, messages transmitted, and generated theory representation. (Some analyses involved several variations on the same protocol.) Elapsed theory generation times using REVERE are in seconds.

Boyer-Moore [BM79], and Ina Jo [LSSE80] verification systems. This line of work was largely based on the Bell-LaPadula security model [BL76], which in the context of a centralized system focuses on subjects' access rights to objects based on security levels. In proving the theorems that expressed security properties of a system or protocol, an expert user would carefully guide the prover, producing lemmas and narrowly directing the proof search to yield results.

More recent theorem-proving efforts have used the HOL [GM93], PVS [ORSvH95], and Isabelle [Pau94] verification systems to express and reason about properties of security protocols. These sophisticated verification systems support specifications in higher-order logic and allow the user to create custom proof strategies and tactics with which the systems can do more effective automated proof search. Though simple lemmas can be proved completely automatically, human guidance is still necessary for most interesting proofs.

We have done limited experiments in applying PVS to the BAN logic as an alternative to theory generation. The encoding of the logic is quite natural, but the proofs are tedious because PVS is often unable to find the right quantified-variable instantiations to apply the BAN logic's rules of inference.

Paulson uses the Isabelle theorem prover to demonstrate a range of security properties in an "inductive approach" [Pau96, BP97]. In this work, he models a protocol as a set of event traces, defined inductively by the protocol specification. He defines rules for deriving several standard message sets from a trace, such as the set of messages (and message fragments) that can be derived from a trace H using only the keys contained in H. Given these definitions, he proposes various classes of properties that can be verified: possibility properties, forwarding lemmas, regularity lemmas, authenticity theorems, and secrecy theorems. Paulson's approach has the advantage of being based on a small set of simple principles, in contrast to the sometimes complex and subtle sets of rules assumed by the BAN logic and related belief logics. It does not, however, provide the same high-level intuition into why a protocol works that the belief logics can. Paulson demonstrates proof tactics that can be applied to prove some lemmas automatically, but significant human interaction still appears to be required.

Brackin recently developed a system [Bra96] within HOL for converting protocol specifications in an extended version of the GNY logic [GNY90] to HOL theories. His system then attempts to prove user-specified properties and certain default properties automatically. This work looks promising; one drawback is that it is tied to a specific logic. Modifying that logic or applying the technique to a new logic would require substantial effort and HOL expertise. In theory generation, the logic can be expressed straightforwardly, and the proving mechanism is independent of the logic.

Like general-purpose theorem proving, theory generation involves manipulation of the syntactic representation of the entity we are verifying. However, by restricting the nature of the logic, unlike machine-assisted theorem proving, we can enumerate the entire theory rather than (with human assistance) develop lemmas and theorems as needed. Moreover, the new method is fast and completely automatic, and thus more suitable for integration into the protocol development process.

6.2 Model Checking

Model checking is a verification technique wherein the system to be verified is represented as a finite state machine, and properties to be checked are expressed as formulas in some temporal logic. The technique involves doing an exhaustive search of the state space to determine whether a given formula holds. Symbolic model checking has been used successfully to verify many concurrent hardware systems, and it has attracted significant interest in the wider verification community due to its high degree of automation and its ability to produce counterexamples when verification fails.

Millen's Interrogator tool could be considered the first model checker for cryptographic protocol analysis [Mil84, KMM94]. It is a Prolog [CM81] system in which the user specifies a protocol as a set of state transition rules, and further specifies a scenario corresponding to some undesirable outcome (e.g., an intruder learns a private key).

Recently, advanced general-purpose model checkers have been applied to protocol analysis with some encouraging results. Lowe used the FDR model checker [Ros94] to demonstrate a flaw in, and then fix, the Needham-Schroeder public key protocol [Low96] and (with Roscoe) the TMN protocol [LR97], and Roscoe used FDR to check noninterference of a simple security hierarchy (high security/low security) [Ros95]. Heintze, Tygar, Wing, and Wong used FDR to check some atomicity properties of NetBill [ST95] and Digicash [CFN88] protocols [HTWW96]. Mitchell, Mitchell, and Stern developed a technique for analyzing cryptographic protocols using Murϕ, a model checker that uses explicit state representation, and, with Shmatikov, have applied it to the complex SSL protocol [MMS97, MSS98]. Marrero, Clarke, and Jha have produced a specialized model checker for reasoning about security protocols, which takes a simple protocol specification as input and has a built-in model of the intruder [CJM98]. Finally, Song [Son99] recently implemented a promising new model checking approach based on the strand model [THG98].

All these model-checking approaches share the limitation that they can consider only a limited number of runs of a protocol—typically one or two—before the number of states of the finite state machine becomes unmanageable. This limitation results from the well-known state explosion problem exhibited by concurrent systems. In some cases it can be worked around by proving that any possible attack must correspond to an attack using at most n protocol runs.

The theory generation technique takes significant inspiration from the desirable features of model checking. Like model checking, theory generation allows "push-button" verification with no lemmas to postulate and no invariants to infer. Whereas model checking achieves this automation by requiring a finite model, theory generation achieves it by requiring a simple logic. Also like model checking, theory generation seeks to provide more interesting feedback than a simple "yes, this property holds" or "I cannot prove this property." In model checking, counterexamples give concrete illustrations of failures, while theory generation offers the opportunity to directly examine and compare theories corresponding to various protocols. By taking advantage of the intuitive belief logics, theory generation can better provide the user with a sense of why a protocol works or how it might be improved; model checking is more of a "black box" in this respect. The two approaches can complement each other, as model checking provides answers without specifying belief interpretations for the protocol, while theory generation presents the user with a higher-level interpretation of the protocol's effects.

6.3 Hybrid Approaches

Meadows' NRL Protocol Analyzer is perhaps the best known tool for computer-assisted security protocol analysis [Mea94]. It is a semi-automated tool that takes a protocol description and a specification of some bad state, and produces the set of states that could immediately precede it. In a sense the Analyzer represents a hybrid of model checking and theorem proving approaches: it interleaves brute-force state exploration with the user-guided derivation of lemmas to prune the search space. It has the notable advantages that it can reason about parallel protocol run attacks and that it produces sample attacks, but it sometimes suffers from the state explosion problem and it requires significant manual guidance. It has been applied to a number of security protocols, and continuing work has increased the level of automation [Mea98]. Theory generation, however, offers greater automation, the benefits of intuitive belief logics, and protocol comparison abilities not available in the Analyzer.

7 Summary, Future Work, and Conclusions

7.1 Summary of Results and Contributions

Theory generation is a new general-purpose technique for performing automated verification. In this paper, we described a simple algorithm (TG_ℓ) for producing finite representations of theories. This algorithm can be applied to any logic in the ℓ_{RW} class that also meets the preconditions in Definition 10. The theory representations produced by TG_ℓ are well suited to direct comparison since they are canonical, and they can be used in an efficient decision procedure. We have proved that the TG_ℓ algorithm and this decision procedure terminate and are correct. We implemented the algorithm in our REVERE system.

We applied theory generation to verify properties about over a dozen security protocols. We expressed the protocols and their desired properties in terms of four different belief logics, including our new RV logic. Using REVERE, we reproduced published protocol analyses using these belief logics, and in some cases we exposed errors in the earlier analyses. In practice, the theory generation process completes quickly (in seconds or minutes), and the theory representations generated are consistently of manageable size.

Our work has demonstrated the utility of theory generation for analyzing security protocols, but this is only a start; further investigation will tell whether it can yield similar benefits in other domains.

7.2 Future Work on Theory Generation

The TG_ℓ algorithm itself could be enhanced, or its preconditions relaxed, in a variety of ways. We consider the termination guarantees first.

The purpose for most of the preconditions is to ensure that TG_ℓ will always terminate, but there is a tradeoff between making the preconditions easy to check and allowing as many logics as possible. The preconditions given in Definition 10 are sufficient but not necessary to ensure termination. We could replace them with the simple condition that each S-rule application must produce a formula no larger than any of the formulas used (perhaps through G-rules) to satisfy its premises. This imposes a considerable burden of proof, but it is a more permissive precondition than the one we use. Furthermore, while it is sufficient to ensure that a finite theory representation exists, it is not sufficient to ensure termination, as we must also prove that each S-rule application attempt must terminate. In some cases, this extra effort may be justified by the increased flexibility.

In practice we may sometimes be fairly confident that a suitable pre-order exists but not want to go through the trouble of producing it. We can skip specifying the pre-order if we are willing to accept the risk of non-termination. Correctness will not be sacrificed, so if the algorithm terminates, we can be sure it has generated the right result.

As an alternative approach to ensuring termination, we could draw on existing research in automatic termination analysis for Prolog—for instance the approach proposed by Lindenstrauss and Sagiv [LS97]—to check whether a set of rules will halt. This would require either adjusting these methods to take account of our use of rewrites, or perhaps encoding the TG_ℓ algorithm itself as a Prolog program whose termination could be analyzed in the context of a fixed set of rules and rewrites.

To improve the performance of TG_ℓ, we could introduce a special case to handle associative-commutative rewrites efficiently, using known techniques for associative-commutative unification. The general-purpose unification modulo equalities implemented for REVERE has acceptable performance for the examples we ran, but it could become expensive when formulas or rules include long sequences. We might also get better performance in searching for formulas matching a given pattern by adapting the Rete algorithm used in some AI systems [For82].

The TG_ℓ algorithm could be modified quite easily to keep track of the proof of each derived formula. This information could prove useful in providing feedback to the user when the verification of some property fails; we could, for instance, automatically fill in "missing" side conditions in an attempt to push the proof through, and then display a proof tree with the trouble spots highlighted. The proofs could also be fed to an independent verifier to double-check the TG_ℓ results.

Thinking further afield, we might consider extensions such as providing theory representations other than the (R, R') representations described in Definition 3, or even representations other than sets of formulas in the target logic. These alternative theory representations might prove necessary in applying theory generation to domains other than security protocols.

7.3 Closing Remarks

The community does not yet have a complete solution to security protocol verification; perhaps that is an unattainable goal. However, today we can provide substantial support for the design and analysis of these protocols, and potential sharing of information among different tools, e.g., via CAPSL [Mil97]. A thorough protocol design process should start with adherence to principles and guidelines such as Abadi and Needham's [AN96]. The designers could apply theory generation with RV or other belief logics to prove that the protocol meets its goals and make explicit the assumptions on which the protocol depends. They could use symbolic model checkers to generate attack scenarios. With interactive, automated theorem proving systems, they could demonstrate that the underlying cryptography meets its requirements, and make the connection between the protocol's behavior and that of the system in which it is used. Finally, the proposed protocol could be presented for public review, so that others might independently apply their favorite formal and informal methods. Each step in this process focuses on some level of abstraction and emphasizes some set of properties, in order to build confidence in the protocol and the system.

References

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols.

IEEE Transactions on Software Engineering, 22(1):6–15, January 1996.

[BAN89] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical Report SRC-39, DEC SRC, 1989.

[BAN90] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[Ben86] Jon Bentley. Little languages. *Communications of the ACM*, 29(8):711–721, 1986.

[BL76] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA, March 1976.

[BM79] R. S. Boyer and J. S. Moore. *A Computational Logic*. ACM monograph series. Academic Press, New York, 1979.

[BP95] Antoon Bosselaers and Bart Preneel, editors. *Integrity primitives for secure information systems: final report of RACE Integrity Primitives Evaluation RIPE-RACE 1040*, volume 1007 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1995.

[BP97] G. Bella and L. C. Paulson. Using Isabelle to prove properties of the Kerberos authentication system. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997.

[Bra96] Stephen H. Brackin. A HOL extension of GNY for automatically analyzing cryptographic protocols. In *Proceedings of the Ninth IEEE Computer Security Foundations Workshop*, pages 62–75, June 1996.

[CFN88] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Proceedings of Advances in Cryptology—CRYPTO '88*, pages 319–327, 1988.

[CJM98] Edmund Clarke, Somesh Jha, and Will Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[CM81] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.

[DS81] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[FKK96] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol, version 3.0. IETF Internet Draft `draft-ietf-tls-ssl-version3-00.txt`, available at `http://home.netscape.com/products/security/ssl`, November 1996.

[For82] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[GM93] M. Gordon and T. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, Cambridge, UK, 1993.

[GMT+80] Susan L. Gerhart, David R. Musser, D. H. Thompson, D. A. Baker, R. L. Bates, R. W. Erickson, R. L. London, D. G. Taylor, and D. S. Wile. An overview of AFFIRM: A specification and verification system. In S. H. Lavington, editor, *Proceedings of IFIP Congress 80*, pages 343–347, Tokyo, Japan, October 1980. North-Holland.

[GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proc. IEEE Symposium on Security and Privacy*, pages 234–248, May 1990.

[Gro99] IETF Secure Shell (secsh) Working Group. Secure shell (secsh) charter. Available at `http://www.ietf.org/html.charters/secsh-charter.html`, March 1999.

[HTWW96] Nevin Heintze, Doug Tygar, Jeannette Wing, and Hao-Chi Wong. Model checking electronic commerce protocols. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 147–164, 1996.

[Kai96] Rajashekar Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, 22(5):313–328, May 1996.

[Kin99] Darrell Kindred. Theory generation for security protocols. PhD Thesis CMU-CS-99-130, Carnegie Mellon University, 1999.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[KW94] Volker Kessler and Gabriele Wedel. AUTLOG—an advanced logic of authentication. In *Proceedings of the Computer Security Foundations Workshop VII*, pages 90–99. IEEE Comput. Soc., June 1994.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055. Springer-Verlag, March 1996. Lecture Notes in Computer Science.

[LR97] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.

[LRS79] K. N. Levitt, L. Robinson, and B. A. Silverberg. The HDM handbook, vols. 1–3. Technical report, SRI International, Menlo Park, California, 1979.

[LS97] Naomi Lindenstrauss and Yehoshua Sagiv. Automatic termination analysis of logic programs. In *Proc. 14th International Conference on Logic Programming*, 1997.

[LSSE80] R. Locasso, J. Scheid, D. V. Schorre, and P. R. Eggert. The Ina Jo reference manual. Technical Report TM-(L)-6021/001/000, System Development Corporation, Santa Monica, California, 1980.

[Mea94] Catherine Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of the Seventh IEEE Computer Security Foundations Workshop*, pages 84–89, June 1994.

[Mea98] Catherine Meadows. Using the NRL protocol analyzer to examine protocol suites. In *Proc. Workshop on Formal Methods and Security Protocols*, Indianapolis, June 1998.

[Mil84] Jonathan K. Millen. The Interrogator: a tool for cryptographic protocol security. In *Proc. IEEE Symposium on Security and Privacy*, April 1984.

[Mil97] Jonathan K. Millen. CAPSL: Common authentication protocol specification language. available at `http://www.jcompsec.mews.org/capsl/`, July 1997.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proc. IEEE Symposium on Security and Privacy*, pages 141–151, May 1997.

[MNSS87] S. P. Miller, C. Neuman, J. I. Schiller, and J. H. Saltzer. *Kerberos authentication and authorization system*, chapter Sect. E.2.1. MIT, Cambridge, Massachusetts, July 1987.

[MSS98] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proc. of the Seventh USENIX Security Symposium*, pages 257, 201–215, January 1998.

[NN93] P. Nivela and R. Nieuwenhuis. Saturation of first-order (constrained) clauses with the Saturate system. In *Proceedings of the Fifth International Conference on Rewriting Techniques and Applications*, pages 436–440, June 1993.

[NS78] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[ORSvH95] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Pau96] Lawrence C. Paulson. Proving properties of security protocols by induction. Technical report, University of Cambridge, December 1996.

[Plo72] G. D. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[Ros94] A. W. Roscoe. Model-checking CSP. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*. Prentice/Hall, 1994.

[Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symposium on Security and Privacy*, pages 114–127, 1995.

[RS96] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. IETF Internet Draft `draft-ietf-wts-shttp-03.txt`, available at `http://www.eit.com/projects/s-http`, July 1996.

[Son99] Dawn Xiaodong Song. Athena, a new efficient automatic checker for security protocol analysis. In *Proceedings of the 12th Security Foundations Workshop*, June 1999.

[ST95] Marvin Sirbu and J. D. Tygar. Netbill: An internet commerce system optimized for network delivered services. In *Digest of Papers for COMPCON'95: Technologies for the Information Superhighway*, pages 20–25, March 1995.

[SvO94] P. F. Syverson and P. C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Symp. on Security and Privacy*, pages 14–28, 1994.

[THG98] P. Thayer, J.C. Hertzog, and J.D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, 1998.

[VCP+95] Manuela Veloso, Jaime Carbonell, Alicia Perez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Theoretical and Experimental Artificial Intelligence*, 7(1), 1995.

[VM96] Visa and MasterCard. Secure Electronic Transaction (SET) specification. Available at `http://www.visa.com/cgi-bin/vee/nt/ecomm/set/setprot.html`, June 1996.

[WL92a] Thomas Y. C. Woo and Simon S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, January 1992.

[WL92b] Thomas Y. C. Woo and Simon S. Lam. 'Authentication' revisited. *Computer*, 25(3):10, March 1992.

Appendix A: TG_ℓ Proofs

This appendix contains the full proofs of correctness and termination for the TG_ℓ algorithm.

A.1 Correctness

The TG_ℓ algorithm is intended to produce a theory representation in the (R, R') form (see Section 2.2), where the "preferred" rules R' are exactly the S-rules. To prove that it does this, we need a few lemmas describing the behavior of certain components of the TG_ℓ algorithm. These lemmas assume that the various functions always terminate; the termination proofs appear in the next section.

We present the following claims regarding the *unify* function without proof, since it is a standard building block and we have not described its implementation in detail.

Claim 1 (*unify* soundness) *If ϕ_1 and ϕ_2 are formulas of ℓ_{RW}, and σ is a substitution such that*

$$\sigma \in \mathit{unify}(\phi_1, \phi_2)\ ,$$

then

$$\{\phi_1\} \vdash_W \sigma\phi_2$$

and

$$\{\phi_2\} \vdash_W \sigma\phi_1\ .$$

Claim 2 (*unify* completeness) *Let ϕ_1 and ϕ_2 be formulas of ℓ_{RW} that share no variables, and let σ be a substitution, such that*

$$\{\phi_1\} \vdash_W \sigma\phi_2$$

or

$$\{\phi_2\} \vdash_W \sigma\phi_1\ .$$

There exists a substitution, σ', such that

$$\sigma' \in \mathit{unify}(\phi_1, \phi_2)$$

and σ is an extension of σ'. (That is, there exists σ'' such that $\sigma = \sigma'' \circ \sigma'$).

This theorem demonstrates the soundness of $\mathit{backward\_chain}$: that each substitution it returns can be applied to the given goals to yield provable formulas.

Theorem 1 (*backward_chain* soundness) *Let Φ and V be sets of formulas of ℓ_{RW}, let Γ be a set of formulas of ℓ_{RW}, and let σ be a substitution, such that*

$$\sigma \in \mathit{backward\_chain}(\Phi, \Gamma, V) \; .$$

Then, for every $\phi \in \Phi$,

$$\Gamma \vdash_{GW} \sigma\phi \; .$$

Proof: This proof is by induction on the total number of recursive invocations of $\mathit{backward\_chain}$. We assume that any invocation of $\mathit{backward\_chain}$ that causes fewer than n recursive calls satisfies the theorem, and show that the result holds for n recursive calls as well.

In order for $\mathit{backward\_chain}$ to return σ, it must be the case that either Γ is empty and σ is the identity (in which case the result follows trivially), or $\sigma = \sigma_2 \circ \sigma_1$, where

$$\begin{aligned}
&\Phi = \{g\} \cup gs \\
&\sigma_1 \in \mathit{backward_chain_one}(g, \mathit{known}, \mathit{visited}) \\
&\sigma_2 \in \mathit{backward_chain}(\mathit{apply_subst}(\sigma_1, gs), \mathit{known}, \mathit{visited}) \; .
\end{aligned}$$

Examining $\mathit{backward\_chain\_one}$, we see that σ_1 can arise in two ways: from $\mathit{regular\_substs}$ or from $\mathit{grule\_substs}$. We will show that in each case,

$$\Gamma \vdash_{\mathrm{GW}} \sigma_1 g \; .$$

Case 1: σ_1 comes from $\mathit{regular\_substs}$. There must exist a formula $F \in \Gamma$, and a substitution σ_1', such that

$$\begin{aligned}
&\sigma_1 = \sigma_1' \circ \sigma_r \\
&\sigma_1' \in \mathit{unify}(\sigma_r g, F)
\end{aligned}$$

By Claim 1 (unify soundness), we have

$$\Gamma \vdash_{\mathrm{GW}} \sigma_1'(\sigma_r g) \; ,$$

so

$$\Gamma \vdash_{\mathrm{GW}} \sigma_1 g$$

and this case is done.

Case 2: σ_1 comes from $\mathit{grule\_substs}$. There must exist a G-rule, R, and a substitution σ_1', such that

$$\begin{aligned}
&\sigma_1 = \sigma_1' \circ \sigma_r \\
&\sigma_1' \in \mathit{reverse_apply_grule}(R, \sigma_r g, \Gamma, \ldots)
\end{aligned}$$

(We ignore the $\mathit{visited}$ argument, since it is irrelevant to soundness.) Following into $\mathit{reverse\_apply\_grule}$, we find that

$$\begin{aligned}
&\sigma_1' = \sigma_4 \circ \sigma_3 \\
&\sigma_3 \in \mathit{unify}(\sigma_r g, \mathit{conclusion}(R)) \\
&\sigma_4 \in \mathit{backward_chain}(\mathit{apply_subst}(\sigma_3, \mathit{premises}(R)), \Gamma, \ldots)
\end{aligned}$$

Claim 1 (unify soundness) implies that

$$\{\mathit{conclusion}(R)\} \vdash_{\mathrm{W}} \sigma_3(\sigma_r g) \; .$$

By the induction assumption, for every P in $\mathit{premises}(R)$,

$$\Gamma \vdash_{\mathrm{GW}} \sigma_4(\sigma_3 P) \; .$$

We can rename variables in these premise-proofs using σ_r, then combine them and add an application of the G-rule, R, with the substitution $\sigma_1' \circ \sigma_r$, to get

$$\Gamma \vdash_{\text{GW}} \sigma_1' \sigma_r g \ .$$

This simplifies to

$$\Gamma \vdash_{\text{GW}} \sigma_1 g \ ,$$

so Case 2 is done.

We now return to *backward_chain*, armed with the knowledge that $\sigma_1 g$ has a proof from Γ. By adding instantiation steps, we can convert this proof to a proof of $\sigma_2\sigma_1 g$, so we have

$$\Gamma \vdash_{\text{GW}} \sigma_2(\sigma_1 g)$$

We can apply the induction assumption to the recursive *backward_chain* invocation, yielding

$$\Gamma \vdash_{\text{GW}} \sigma_2(\sigma_1 G)$$

for every $G \in gs$. Since $\Phi = \{g\} \cup gs$, and $\sigma = \sigma_2 \circ \sigma_1$, we have shown that for every $\phi \in \Phi$, there exists a proof of

$$\Gamma \vdash_{\text{GW}} \sigma\phi \ ,$$

using no S-rules. ■

Now we show the dual of Theorem 1, the completeness of *backward_chain*; that is, that it returns every most-general substitution under which the goals are provable.

Theorem 2 (*backward_chain* completeness) *Let Γ and V be sets of formulas, let Φ be a set of formulas ($\{\phi_1, \ldots, \phi_n\}$), let σ be a substitution, and let $\mathcal{P}_1 \ldots \mathcal{P}_n$ be proofs (using no S-rules), such that for $1 \leq i \leq n$,*

$$\Gamma \overset{\mathcal{P}_i}{\vdash}_{GW} \sigma\phi_i$$

and the proofs, $\mathcal{P}_i$, contain no rewrites of formulas in V. Then, if backward_chain terminates, there exists a substitution, σ', such that

$$\sigma' \in backward\_chain(\Phi, \Gamma, V)$$

and σ is an extension of σ'.

Proof: We prove this theorem by induction on the total number of G-rule applications in $\mathcal{P}_1 \ldots \mathcal{P}_n$.

Without loss of generality, we assume that the proofs, $\mathcal{P}_1 \ldots \mathcal{P}_n$, have no "sharing." That is, for any proof line that is used as a premise more than once, we duplicate the proof prefix ending with that line to eliminate the sharing. To carry out the induction, we now assume that the theorem holds when $\mathcal{P}_1 \ldots \mathcal{P}_n$ contain a total of fewer than n G-rule applications, and show that it holds when there are n G-rule applications.

In the case where Γ is empty, the theorem holds trivially, so assume Γ is non-empty. Let ϕ_i be the first goal selected by *choose_goal*. There are two cases to consider, depending on whether $\mathcal{P}_i$ contains any G-rule applications.

Case 1: $\mathcal{P}_i$ contains no G-rule applications.

Looking at the call to *backward_chain_one*, we can see that the $\hat{\phi} \in visited$ check will not be triggered since ϕ_i must be in the proof $\mathcal{P}_i$, and thus no rewrite of it can appear in V. Since σ_r is just a renaming, there exists some σ' such that

$$\sigma = \sigma' \circ \sigma_r \ .$$

Thus,

$$\Gamma \vdash_{\text{GW}} \sigma' \sigma_r \phi_i$$

and Claim 2 implies there exists some σ'' for which

$$\sigma'' \in \bigcup_{F \in known} unify(\phi_r, F)$$

and σ' is an extension of σ''. The set returned by $backward\_chain\_one$ will thus include $\sigma'' \circ \sigma_r$, of which $\sigma' \circ \sigma_r$ (that is, σ) is an extension. Therefore, $backward\_chain$ will return as one of its results, $\sigma_2 \circ \sigma_1$, where σ is an extension of σ_1, and

$$\sigma_2 \in backward\_chain(apply\_subst(\sigma_1, gs), \Gamma, V) \ .$$

We have thus reduced this case to a case with the same total number of G-rule applications and one fewer formula in Φ, so without loss of generality we can assume the second case.

Case 2: $\mathcal{P}_i$ contains at least one G-rule application.

The recursive call to $backward\_chain$ passes a (partially instantiated) subset of Φ (all but ϕ_i); since $\mathcal{P}_i$ contains some G-rule applications, the proofs for this subset must contain fewer G-rule applications than $\mathcal{P}_1 \ldots \mathcal{P}_n$, so we can apply the induction hypothesis. This implies that, as long as σ is an extension of some σ_1, the theorem holds. It remains only to demonstrate this.

In $backward\_chain\_one$, again, the $\hat{\phi} \in visited$ check will not be triggered for the same reason as in Case 1. As in Case 1, there exists some σ' such that

$$\sigma = \sigma' \circ \sigma_r \ .$$

Let R be the last G-rule applied in the proof, $\mathcal{P}_i$. If we can prove that

$$reverse\_apply\_grule(R, \phi_r, \Gamma, V \cup \{\hat{\phi}\})$$

returns some σ'' of which σ' is an extension, then by the argument in Case 1, $backward\_chain\_one$ will return a substitution of which σ is an instance, so the theorem will hold.

Since σ' is just a variable-renaming followed by σ, we can transform the proof of $\sigma\phi_i$, $\mathcal{P}_i$, into a proof of $\sigma'\phi_i$, called $\mathcal{P}'_i$, by simple renaming. From $\mathcal{P}'_i$, we can extract a proof of each premise of its last G-rule application, and also a proof of $\sigma'\phi_i$ from the G-rule conclusion. By Claim 2, σ' is an extension of some σ_3 that will be returned by $unify(\sigma_r\phi_i, conclusion(R))$. The proofs of R's premises will not contain any rewrites of V, since these proofs come from $\mathcal{P}'_i$, and we will further assume they contain no rewrites of ϕ_i. (If they did, the application of R could be eliminated from $\mathcal{P}_i$, so there is no loss of generality from this assumption.) Since we have proofs of R's premises (instantiated by σ'), which have fewer total G-rule applications than the original $\mathcal{P}_i$, and since these proofs contain no rewrites of formulas in the (expanded) $visited$ set, we can apply the induction hypothesis and find that for some σ_4 returned by the $backward\_chain$ call, σ' is an extension of $\sigma_4 \circ \sigma_3$, which will be returned by $reverse\_apply\_grule$. This is the final result we required to complete the proof of the theorem. ∎

Now we can prove the soundness and completeness of the $closure$ function, the heart of the TG_ℓ algorithm.

Theorem 3 ($closure$ soundness) *Let Γ and Γ' be sets of formulas of ℓ_{RW}. For any formula, F, where*

$$F \in closure(\Gamma', \Gamma) \ ,$$

there exists a proof, $\mathcal{P}$, of $\Gamma \cup \Gamma' \vdash F$, in which the last rule application (if any) is of an S-rule.

Proof: The proof is by induction on the number of recursive calls to $closure$. If there are no such calls, Γ' (the fringe) must be empty, so Γ is returned, and the theorem is trivially satisfied. Otherwise, $closure$ is called recursively with the formulas in *fringe* added to Γ and *fringe*$'$ becomes the new fringe. If we can show that all of the *fringe*$'$ formulas have proofs from Γ of the appropriate form, then we can apply the induction assumption and the theorem is proved.

For every S-rule, R, $closure$ calls $apply\_srule(R, \Gamma \cup \Gamma')$, which will return R's conclusion, instantiated by σ, where σ comes from the $backward\_chain$ result. By Theorem 1 ($backward\_chain$ soundness), for each premise, P, of R, there exists a proof of

$$\Gamma \cup \Gamma' \vdash \sigma P$$

We can concatenate these proofs, followed by an application of the S-rule, R, to yield a proof of σC (where C is R's conclusion). Furthermore, this proof has no rule applications following the application of R, so the proof is of the appropriate form. It is therefore safe to add

$$make\_canonical(apply\_subst(\sigma, conclusion(R)))$$

to the fringe, since $make\_canonical$ applies only rewrites, and not G- or S-rules. ∎

To express the next theorem, we introduce a notion of *partial theory representations*:

Definition 13 *If, for any formula, ϕ, and proof, $\mathcal{P}$, such that $\mathcal{P}$ has only one S-rule application and no other rule applications following it, and where*

$$\Gamma \overset{\mathcal{P}}{\vdash} \phi\ ,$$

it is also the case that

$$(\Gamma' \cup \Gamma) \vdash_W \phi\ ,$$

then we call the ordered pair, $\langle \Gamma', \Gamma \rangle$, a partial theory representation.

Note that if T is a theory representation, then $\langle \{\}, T \rangle$ is a partial theory representation. The $closure$ function takes a partial theory representation and produces a theory representation:

Theorem 4 ($closure$ completeness) *Let T and fringe be sets of formulas (of ℓ_{RW}), such that $\langle fringe, T \rangle$ is a partial theory representation. For any formula, ϕ, and proof, $\mathcal{P}$, whose last rule application is an S-rule application, where*

$$(T \cup fringe) \overset{\mathcal{P}}{\vdash} \phi\ ,$$

there exists some ϕ', where (assuming closure terminates)

$$\phi' \in closure(fringe, T)\ ,$$

such that

$$\phi' \vdash_W \phi\ .$$

Proof: The proof is by induction on the number of recursive calls to $closure$, which is guaranteed to be finite since we assume $closure$ terminates.

If there are no recursive calls to $closure$, then *fringe* is empty, and $closure$ returns just T, which by Definition 13 must satisfy

$$T \vdash_{\mathrm{W}} \phi\ ,$$

so this case is done.

If fringe is non-empty, then the function returns the result of

$$\mathit{closure}(\mathit{fringe}', T') \ .$$

Since

$$T' \supset (T \cup \mathit{fringe}) \ ,$$

the induction hypothesis will yield the desired result if we can prove that

$$\langle \mathit{fringe}', T' \rangle$$

is a partial theory representation.

Let $\phi^\star$ be a formula as given in Definition 13, where

$$T' \overset{\mathcal{P}^\star}{\vdash} \phi^\star \ ,$$

and $\mathcal{P}^\star$ has exactly one S-rule application, which is its last rule application. Let R be the last S-rule applied in $\mathcal{P}^\star$, let C be R's conclusion, and let σ be the substitution under which R was applied. From Theorem 3 ($\mathit{backward\_chain}$ completeness), it follows that σ is an extension of some substitution returned by

$$\mathit{backward\_chain}(\mathit{premises}(R), T', \{\}) \ ,$$

and thus

$$\mathit{apply\_srule}(R, T')$$

will return some formula of which σC is an instance. Since $\mathit{make\_canonical}$ only transforms one formula into another that is equivalent modulo rewrites, we get

$$\mathit{make\_canonical}(\mathit{apply\_srule}(R, T')) \vdash_{\mathrm{w}} \sigma C$$

and finally, this implies that

$$\mathit{fringe}' \cup T' \vdash_{\mathrm{w}} \sigma C \ . \tag{1}$$

Now, returning to the proof, $\mathcal{P}^\star$, since no more rules are applied after R, it follows that

$$\sigma C \vdash_{\mathrm{w}} \phi \ .$$

This, combined with (1), gives the result we need:

$$\mathit{fringe}' \cup T' \vdash_{\mathrm{w}} \phi \ .$$

This completes the induction. ■

We can now prove the claim made at the beginning of this section, which corresponds to the following theorem:

Theorem 5 (**TG_ℓ Correctness**) *If Γ is a set of mostly-ground formulas of ℓ_{RW}, and S_rules, G_rules, $Rewrites$, and $\preceq$ meet the TG_ℓ algorithm preconditions given in Definition 10, then if*

$$\mathit{theory\_gen}(\Gamma, S\_rules, G\_rules, \mathit{Rewrites}, \preceq)$$

terminates, it returns an (R, R') representation of the theory induced by Γ, where R' is the set of S-rules, and the equivalence used is equivalence modulo rewrites and variable renaming.

Proof: First we prove that every formula returned by $theory\_gen$ is in the (R, R') representation. Let F be a formula returned by $theory\_gen$. It must be the case that

$$F \in closure(make\_canonical(\Gamma), \{\}) \ ,$$

and so by Theorem 2 ($closure$ soundness), there exists a proof, $\mathcal{P}$, such that

$$make\_canonical(\Gamma) \overset{\mathcal{P}}{\vdash} F \ ,$$

and where the last rule application in $\mathcal{P}$ is of an S-rule. Since $make\_canonical$ only transforms by rewrites, we know that

$$make\_canonical(\Gamma) \overset{\mathcal{P}}{\vdash} F \quad \Longrightarrow \quad \Gamma \overset{\mathcal{P}'}{\vdash} F$$

and furthermore, that since the last rule applied in $\mathcal{P}$ is an S-rule, the same is true of $\mathcal{P}'$. Lastly, since every formula returned by $closure$ has been canonicalized, no two formulas returned by $theory\_gen$ are equivalent modulo rewrites and variable renamings. Therefore, every formula returned by $theory\_gen$ is in the (R, R') representation.

It remains to prove that any formula in the (R, R') representation is returned by $theory\_gen$. Let F be a formula and $\mathcal{P}$ a proof whose last rule applied is an S-rule, such that

$$\Gamma \overset{\mathcal{P}}{\vdash} F \ .$$

By the same argument made above, it follows that

$$make\_canonical(\Gamma) \overset{\mathcal{P}'}{\vdash} F \ ,$$

where $\mathcal{P}'$ has the same property. By Theorem 4 ($closure$ completeness), there exists some F',

$$\phi' \in closure(make\_canonical(\Gamma), \{\}) \ ,$$

such that

$$F' \vdash_{\mathrm{w}} F \ .$$

Therefore, F is equivalent (modulo rewrites and variable renaming) to some formula in the set returned by $theory\_gen$. This completes the correctness proof for the TG_ℓ algorithm. ■

A.2 Termination

The completeness proofs in the previous section assumed that the functions making up the TG_ℓ algorithm always terminated. In this section, we show how the TG_ℓ preconditions ensure this. The proofs below assume the existence of a fixed set of S-rules, G-rules, and rewrites, and a pre-order, $\preceq$, all of which satisfy the TG_ℓ preconditions in Definition 10.

Definition 14 *A formula, F, is* size-bounded *by a finite set of formulas, Γ, when, for any substitution, σ, there exists $G \in \Gamma$ such that*

$$\sigma F \preceq G \ .$$

Note that if F is mostly-ground then F is size-bounded by $\{F\}$.

Lemma 3 *If ϕ_1 is size-bounded by Γ, and $\phi_2 \preceq \phi_1$, then ϕ_2 is size-bounded by Γ.*

Proof: Since $\phi_2 \preceq \phi_1$, we can use pre-order condition P2 ($\preceq$ preserved under substitution) to show that, for any σ,

$$\sigma\phi_2 \preceq \sigma\phi_1 \ .$$

Since ϕ_1 is size-bounded by Γ, there exists some $G \in \Gamma$ such that

$$\sigma\phi_1 \preceq G \ .$$

Applying the transitive property of pre-orders, we get

$$\sigma\phi_2 \preceq G \ ,$$

so ϕ_2 is size-bounded by Γ. ■

Lemma 4 *If the formula, F, is size-bounded by Γ, then for any formula, F', such that*

$$\{F\} \vdash_{\mathrm{W}} F' \ ,$$

F' is also size-bounded by Γ.

Proof: We show that size-boundedness is preserved by instantiation and by rewriting, the only transformations possible in the proof of

$$\{F\} \vdash_{\mathrm{W}} F' \ .$$

Let σ be some substitution. Since F is size-bounded by Γ, there exists some $G \in \Gamma$ such that for any substitution, σ',

$$\sigma'\sigma F \preceq G \ ,$$

so σF is size-bounded by Γ. Let $S = T$ be a rewrite, and let F' be the result of applying that rewrite to F. Rewrites are required to be size-preserving, so $S \preceq T$ and $T \preceq S$. By pre-order condition P1, this implies $F' \preceq F$, and we can apply Lemma 3 to see that F' is size-bounded by Γ. ■

Lemma 5 *If F is size-bounded by Γ, then $make\_canonical(F)$ is size-bounded by Γ.*

Proof: Since $make\_canonical$ only transforms by rewrites and variable renaming, it follows directly from Lemma 4 that $make\_canonical(F)$ is size-bounded by Γ if F is. ■

Lemma 6 *For any finite set of formulas, Γ, there are finitely many formulas that are both size-bounded by Γ and canonical with respect to variable-renaming.*

Proof: By Definition 11, any formula, ϕ, that is size-bounded by Γ must satisfy $\phi \preceq G$ for some $G \in \Gamma$. By pre-order condition P3, since Γ is finite, there are finitely many such formulas, ϕ, modulo variable renaming.

Lemma 7 *If ϕ is size-bounded by the set of formulas, Γ', and R is a G-rule, then*

$$reverse\_apply\_grule(R, \phi, \Gamma, V)$$

will always pass a set of formulas, Φ, to $backward\_chain$, where all formulas in Φ are size-bounded by Γ'. (Γ need have no relation to Γ'; in particular Γ' may contain larger formulas than Γ.)

Proof: Let C be R's conclusion. For any σ_3, such that

$$\sigma_3 \in \mathit{unify}(\phi, C) \ ,$$

Claim 1 (unify soundness) implies that

$$\{\phi\} \vdash_{\mathrm{w}} \sigma_3 C \ .$$

Since ϕ is size-bounded by Γ', it follows that $\sigma_3 C$ is also size-bounded by Γ'. From the G-rule definition, for all premises, P, of R,

$$P \preceq C \ ,$$

and so

$$\sigma_3 P \preceq \sigma_3 C \ .$$

By Lemma 3, $\sigma_3 P$ must be size-bounded by Γ'. The call to $\mathit{backward\_chain}$ sends only formulas of this form. ∎

Lemma 8 *If all formulas in Γ are size-bounded by Γ', and ϕ matches no G-rule conclusion, then for every σ such that*

$$\sigma \in \mathit{backward\_chain\_one}(\phi, \Gamma, V) \ ,$$

$\sigma\phi$ is size-bounded by Γ'.

Proof: Since ϕ matches no G-rule conclusion, neither will the renamed version, ϕ_r, and so $\mathit{grule\_substs}$ will be empty. By Claim 1 (unify soundness), for every σ' such that

$$\sigma' \in \mathit{unify}(\phi_r, F)$$

where $F \in \Gamma$, we know that

$$\{F\} \vdash_{\mathrm{w}} \sigma' \phi_r \ .$$

Therefore, by Lemma 4, since Γ is size-bounded by Γ', so is $\sigma'\phi_r$. Finally, the substitutions returned by $\mathit{backward\_chain\_one}$ are $\sigma' \circ \sigma_r$, and

$$\sigma' \phi_r = \sigma' \sigma_r \phi \ ,$$

so $(\sigma' \circ \sigma_r)\phi$ is size-bounded by Γ'. ∎

Lemma 9 *If Γ is size-bounded by Γ', and there exists some $\phi \in \Phi$ such that ϕ matches no G-rule conclusion, then for every σ such that*

$$\sigma \in \mathit{backward\_chain}(\Phi, \Gamma, V) \ ,$$

$\sigma\phi$ is size-bounded by Γ'.

Proof: Each recursive call to $\mathit{backward\_chain}$ applies another substitution to the remaining goals, and substitution cannot cause a formula to match a G-rule conclusion if it did not already, and it also preserves size-boundedness. Therefore, in some call to $\mathit{backward\_chain}$, the chosen goal, g, will match no G-rule conclusions and will be size-bounded by Γ'. By Lemma 8, every σ_1 substitution such that

$$\sigma_1 \in \mathit{backward\_chain\_one}(g, \Gamma, V)$$

will give $\sigma_1 g$ size-bounded by Γ'. The substitutions returned by the original $\mathit{backward\_chain}$ invocation are extensions of these σ_1 substitutions, so $\sigma\phi$ will be size-bounded by Γ'. ∎

Lemma 10 *If ϕ is size-bounded by Γ', then $backward\_chain\_one(\phi, \Gamma, V)$ will always pass the formula ϕ_r to $reverse\_apply\_grule$, where ϕ_r is also size-bounded by Γ'.*

Proof: This follows directly from the definition of size-bounded, since ϕ_r is the result of a substitution applied to ϕ. ■

Lemma 11 *If Γ is size-bounded by Γ', and Φ is the set of premises of some S-rule, then $backward\_chain(\Phi, \Gamma, V)$ will terminate.*

Proof: Since $choose\_goal$ always selects goals that match no G-rule conclusions first, $backward\_chain$ will satisfy all the primary premises in Φ before examining side-conditions. For each primary premise, $backward\_chain\_one$ will clearly terminate since $reverse\_apply\_grule$ will not call $backward\_chain$.

Let σ_P be the accumulated substitution once the primary premises have been satisfied. Since σC is size-bounded by Γ', and each side-condition, P, satisfies

$$P \preceq C ,$$

it follows from Lemma 3 that for each side-condition, σP is size-bounded by Γ'. A simple induction shows that the recursive call to $backward\_chain$ in $backward\_chain$ itself will preserve this property, and reduces the number of goals by one, so the only possibly non-terminating call is the one to $backward\_chain\_one$.

The call to $backward\_chain\_one$ passes a formula, ϕ, that is size-bounded by Γ'. By Lemmas 7 and 10, any recursive call made to $backward\_chain$ in $reverse\_apply\_grule$ will use goals also size-bounded by Γ'. Since $backward\_chain\_one$ adds the canonical form of ϕ to the visited set, and terminates if ϕ is already in that set, the recursive nesting depth is bounded by the number of such formulas ϕ that are distinct modulo renaming. Since each such ϕ is size-bounded by Γ, pre-order condition P3 implies that there are a finite number of possible ϕ's. Therefore, this recursion must halt, so $backward\_chain$ will terminate. ■

Lemma 12 *If the formulas in Γ are size-bounded by Γ', and R is an S-rule, then the formulas returned by $apply\_srule(R, \Gamma)$ are size-bounded by Γ'.*

Proof: Let C be the conclusion of the S-rule, R, and let P be a primary premise of R that satisfies

$$C \preceq P .$$

By the S-rule definition, some such P must exist, and by the S/G restriction, P must match no G-rule conclusions. Lemma 9 thus implies that for every substitution, σ, such that

$$\sigma \in backward\_chain(premises(R), \Gamma, \{\}) ,$$

σP is size-bounded by Γ'. By the S-rule definition,

$$C \preceq P ,$$

so by pre-order condition P2,

$$\sigma C \preceq \sigma P ,$$

and since σP is size-bounded by Γ', Lemma 3 implies that σC is size-bounded by Γ'. ■

Lemma 13 *If formulas in Γ are size-bounded by Γ', and R is an S-rule, then $apply\_srule(R, \Gamma)$ will terminate.*

Proof: The call to $backward\_chain$ passes the premises of R and the set Γ, so the antecedent of Lemma 11 is satisfied, and $backward\_chain$ (and thus $apply\_srule$) will terminate. ■

Lemma 14 *In $closure$, if the formulas in $fringe$ and T are size-bounded by Γ, then formulas in $fringe'$ and T' are also size-bounded by Γ.*

Proof: First, T' is clearly size-bounded by Γ since T' is just $fringe \bigcup T$. By Lemma 5 and Lemma 12, for each S-rule, R,

$$make\_canonical(apply\_srule(R, T'))$$

is size-bounded by Γ, and so $fringe'$ is also size-bounded by Γ. ■

Lemma 15 *If formulas in $fringe$ and T are size-bounded by some finite set, Γ, and canonical with respect to rewrites and variable renaming, then $closure(fringe, T)$ will terminate.*

Proof: In each recursive call, the set $fringe \bigcup T$ must grow monotonically, until $fringe$ is empty and $closure$ terminates. By Lemma 13, $apply\_srule(R, T')$ must terminate

By Lemma 14, and the definition of $make\_canonical$, these invariants are preserved:

- Formulas in $fringe \bigcup T$ are size-bounded by Γ.
- Formulas in $fringe \bigcup T$ are canonical with respect to rewrites

Therefore, by Lemma 6, there are finitely many formulas that can ever be added to $fringe \bigcup T$, and so the recursion must terminate. ■

We can now prove the termination theorem for the TG_ℓ algorithm:

Theorem 6 *If the TG_ℓ preconditions hold, then $theory\_gen$ will terminate.*

Proof: Assumptions are mostly-ground, so they are size-bounded by themselves. By Lemma 5, the formulas passed to $closure$ are size-bounded by Γ, and they are also canonical with respect to rewrites and variable renaming, so Lemma 15 implies that $closure$, and thus $theory\_gen$, will terminate. ■