

Skolem Machines

John Fisher*

Department of Computer Science
California State Polytechnic University
Pomona, California, USA
jrfisher@csupomona.edu

Marc Bezem

Department of Computer Science
University of Bergen
Bergen, Norway
bezem@ii.uib.no

Abstract. The Skolem machine is a Turing-complete machine model where the instructions are first-order formulas of a specific form. We introduce Skolem machines and prove their logical correctness and completeness. Skolem machines compute queries for the Geolog language, a rich fragment of first-order logic. The concepts of Geolog trees and complete Geolog trees are defined, and these tree concepts are used to show logical correctness and completeness of Skolem machine computations. The universality of Skolem machine computations is demonstrated. Lastly, the paper outlines implementation design issues using an abstract machine model approach.

Keywords: Finitary geometric logic, Skolem machines, Geolog language, Geolog trees, correctness, completeness, universality.

1. The Geolog Language

Geolog is a language for expressing first-order geometric logic in a format suitable for computations using an *abstract machine*. *Geolog* rules are used as machine instructions for an abstract machine that computes consequences for first-order geometric logic.

Address for correspondence: Please contact either author regarding questions.

*Thanks to M. Bezem and UIB for hosting me in Bergen, March 2008, to work on this paper.

A *Geolog* rule has the general form

$$A_1, A_2, \dots, A_m \Rightarrow C_1 \mid C_2 \mid \dots \mid C_n \quad (1)$$

where the A_i are atomic expressions and each C_j is a conjunction of atomic expressions, $m, n \geq 1$. The left-hand side of a rule is called the *antecedent* of the rule (a conjunction) and the right-hand side is called the *consequent* (a disjunction). All atomic expressions can contain variables.

If $n = 1$ then there is a single consequent for the rule (1), and the rule is said to be *definite*. Otherwise the rule is a *splitting rule* that requires a case distinction (case of C_1 or case of C_2 or ... case of C_n).

The separate cases (disjuncts) C_j must have a conjunctive form

$$B_1, B_2, \dots, B_h \quad (2)$$

where the B_i are atomic expressions, and $h \geq 1$ varies with j . Any free variables occurring in (2) other than those which occurred free in the antecedent of the rule are taken to be existential variables and their scope is this disjunct (2).

As an example, consider the *Geolog* rule

$$s(X, Y) \Rightarrow e(X, Y) \mid \text{dom}(Z), r(X, Z), s(Z, Y).$$

The variables X, Y are universally quantified and have scope covering the entire formula, whereas Z is existentially quantified and has scope covering the last disjunct in the consequent of rule. A fully quantified first-order logical formula representation of this *Geolog* rule would be

$$(\forall X)(\forall Y)[s(X, Y) \rightarrow e(X, Y) \vee (\exists Z)(\text{dom}(Z) \wedge r(X, Z) \wedge s(Z, Y))]$$

Now we come to two special cases of rule forms, the *true* antecedent and the *goal* or *false* consequents. Rules of the form

$$\text{true} \Rightarrow C_1 \mid C_2 \mid \dots \mid C_n \quad (3)$$

are called *factuals*. Here ‘*true*’ is a special constant term denoting the empty conjunction. Factuals are used to express initial information in *Geolog* theories. Rules of the form

$$A_1, A_2, \dots, A_m \Rightarrow \text{goal} \quad (4)$$

are called *goal* rules. Here ‘*goal*’ is a special constant term. A *goal* rule expresses that its antecedent is sufficient (and relevant) for *goal*. Similarly, rules of the form

$$A_1, A_2, \dots, A_m \Rightarrow \text{false} \quad (5)$$

are called *false* rules. Here ‘*false*’ is a special constant term denoting the empty disjunction. A *false* rule expresses rejection of its antecedent.

The constant terms *true*, *goal* and *false* can only appear in *Geolog* rules as just described. All other predicate names, individual constants, and variable names are the responsibility of the *Geolog* programmer.

A *Geolog theory* (or *program*) is a finite set of *Geolog* rules. A theory may have any number of *factuals* and any number of *goal* or *false* rules.

The logical formulas characterized by *Geolog*, and the bottom-up approach to reasoning with those logical formulas, finds its earliest precursor (1920) in a particular paper by Thoralf Skolem [17].

2. Skolem Machines

The rules in *Geolog* theories serve as instructions for an abstract *Skolem machine* (SM). Skolem machines resemble multitape Turing machines and the two machine models have actually the same computational power. See the discussion in Section 1.

An SM starts with one tape, the *initial tape* having *true* written on it. The basic operations of an SM use the *Geolog* rules in the instruction set to

- extend a tape (write logical terms at the end)
- create new tapes (for splitting rules)

The tapes are also called *states*. An SM with more than one tape is said to be in a *disjunctive* state, comprised of multiple separate simple states or tapes.

The basic purpose of a particular SM is to compute its instruction set and to halt when all of its tapes have ‘*goal*’ or ‘*false*’ written on them.

In order to motivate the general definitions for the workings of SM, let us work through a small example. To this end, consider the *Geolog* rulebase (SM instructions) in Figure 1.

```

true => domain(X), p(X).                % #1
p(X) => q(X) | r(X) | domain(Y), s(X,Y). % #2
domain(X) => u(X).                      % #3
u(X), q(X) => false.                   % #4
r(X) => goal.                          % #5
s(X,Y) => goal.                        % #6

```

Figure 1. Sample instructions

The only instruction that applies to the initial tape is instruction #1. The antecedent of the rule matches *true* on the tape, so the tape can be *extended* using the consequent of the rule. In order to extend the tape using *domain(X)*, *p(X)* an instance for the free existential variable *X* is first generated and then substituted, and the resulting terms are written on the tape, as shown in Figure 2.

```

-----
true domain(sk1) p(sk1)
-----

```

Figure 2. After applying rule #1

At this point in machine operation time either of the rules #2 or #3 can apply. The general definition of SM operation does not specify the order, but we will apply applicable rules in top-down order. So, applying instruction #2 we get tape *splitting*, as shown in Figure 3.

Each of the disjuncts in the consequent of rule #2 is used to extend the previous single tape. This requires that the previous tape be copied to two new tapes and then these tapes are extended.

Now, instruction #3 applies to all three tapes, even twice to the last tape, with total result shown in Figure 4.

```

-----
true domain(sk1) p(sk1) q(sk1)
-----
true domain(sk1) p(sk1) r(sk1)
-----
true domain(sk1) p(sk1) domain(sk2) s(sk1,sk2)
-----

```

Figure 3. After applying rule #2

```

-----
true domain(sk1) p(sk1) q(sk1) u(sk1)
-----
true domain(sk1) p(sk1) r(sk1) u(sk1)
-----
true domain(sk1) p(sk1) domain(sk2) s(sk1,sk2) u(sk1) u(sk2)
-----

```

Figure 4. After applying rule #3 four times (!)

Instruction #4 now adds `false` to the top tape, shown in Figure 5.

Now instruction #5 applies to the second tape, and then instruction #6 applies to the third tape, shown in Figure 6.

At this point the SM *halts* because each tape has either the term `goal` or the term `false` written on it.

The SM has effectively computed a proof that the *disjunction*

$$(\exists X)(u(X) \wedge q(X)) \vee (\exists X)r(X) \vee (\exists X)(\exists Y)s(X, Y)$$

is a *logical* consequence of the *Geolog* theory consisting of the first three rules in Figure 1. This is so because every tape of the halted machine either has `q(sk1)`, `u(sk1)` written on it or has `r(sk1)` written on it or else has `s(sk1, sk2)` written on it. Note that the three disjuncts correspond to the *goal* and *false* rules in Figure 1. We will continue a discussion of this example (specifically, the role intended for the *false* rule) later in this section.

The *proof tree* displayed in Figure 7 was automatically drawn by the program whose implementation is described in [6]. The diagram displays the tapes generated by the SM in the form of a directed tree. Notice that the tree splits where the SM would have copied a tape. It is possible to describe an SM using trees rather than multiple tapes; see the next section.

```

-----
true domain(sk1) p(sk1) q(sk1) u(sk1) false
-----
true domain(sk1) p(sk1) r(sk1) u(sk1)
-----
true domain(sk1) p(sk1) domain(sk2) s(sk1,sk2) u(sk1) u(sk2)
-----

```

Figure 5. Goal tape, rule #4

```

-----
true domain(sk1) p(sk1) q(sk1) u(sk1) false
-----
true domain(sk1) p(sk1) r(sk1) u(sk1) goal
-----
true domain(sk1) p(sk1) domain(sk2) s(sk1,sk2) u(sk1) u(sk2) goal
-----

```

Figure 6. After applying rule #5 and then #6, HALTED

DEFINITION OF SKOLEM MACHINE OPERATIONS

- A *Geolog* rule $ANT \Rightarrow CONS$ is *applicable* to an SM tape T , provided that it is the case that all of the terms of ANT can be simultaneously matched against ground terms (no free variables) written on T . (It may be that ANT can be matched against T in more than one way; for example, rule #3 and the third tape of Figure 3.)
- If the rule $ANT \Rightarrow CONS$ is applicable to tape T , then for some matching substitution σ apply σ to $CONS$ and then *expand* tape T using $\sigma(CONS)$.
- In order to *expand* tape T by $\sigma(CONS) = C_1 \mid C_2 \mid \dots \mid C_k$ copy tape T making $k - 1$ new tapes T_2, T_3, \dots, T_k , and then *extend* T using C_1 , extend T_2 using C_2, \dots , and extend T_k using C_k . (No copying if $k = 1$.)
- In order to *extend* a tape T using a conjunction C , suppose that X_1, \dots, X_p are all of the free existential variables in C . Create new constants c_j , $1 \leq j \leq p$ and substitute c_j for X_j in C , obtaining C' , and then write each of the terms of C' on tape T . It is mandatory that the constant is new with respect to the theory and the tape.¹

¹These witnesses are called Skolem constants by some, but we would prefer to view them as *eigenvariables* in the elimination of existential quantification.

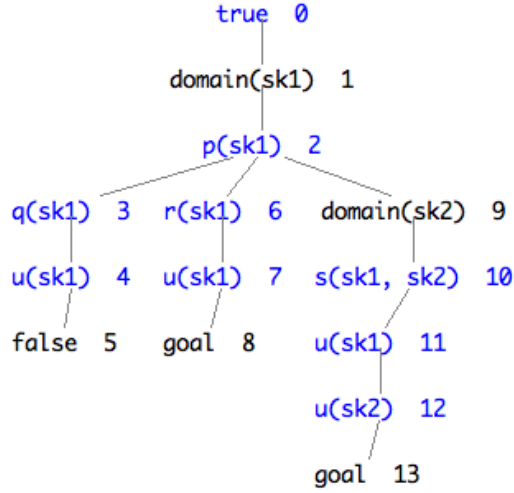


Figure 7. Tree display

Notice that only ground terms ever appear on any SM tape. Thus the matching algorithm does not really need the full power of general term unification. Simple left-to-right term matching suffices.

Given an SM with tapes T_1, \dots, T_t , $t \geq 0$, we say that a particular tape T_i is *saturated* if no applicable instance of a rule leads to new facts.

A tape is *halted* if it is either saturated or contains *goal* or contains *false* (any of which could occur at the same time). An SM is called *halted* if all its tapes are halted, it is *halted successfully* if it is halted with all tapes containing either *goal* or *false*. If a tape of an SM is saturated with neither *goal* nor *false* on it, then this tape actually constitutes a countermodel: all rules are satisfied, they are consistent (by absence of *false*) and yet the goal is false (by absence of *goal*).

The set of terms on any saturated tape that is not successfully halted is said to be a *counter model*.

Suppose that we write a *Geolog* theory in the form

$$T = A \cup G \cup F \quad (6)$$

where A is the *axioms*, G contains all of the affirming *goal* rules and F contains all of the rejecting *false* rules. It is intended that A contains all the rules of the theory other than the *goal* rules and the *false* rules and that A , G , and F are mutually disjoint sets.

The *Geolog query* Q for a *Geolog* theory $T = A \cup G \cup F$ is the disjunctive normal form $Q = C_1 \mid C_2 \mid \dots \mid C_k$ consisting of all of the conjunctions C_i such that either C_i appears as antecedent of one of the *goal* rules (in G) or of one of the *false* rules (in F). As before, the free variables in Q are taken to be existential variables. The scope of a variable X appearing in a particular C_i (within Q) is restricted to C_i .

We say that a *Geolog* theory T *supports* its query Q if there is a successfully halted SM such that each tape satisfies some C_i .

Theorem 1. *If theory T supports its query Q then Q is a logical consequence of the axioms.*

Theorem 2. Suppose that Q is the query for Geolog theory G and that Q is a logical consequence of G . Then G supports Q .

Theorem 1 is proved in Section 3 next, as a corollary to a general characterization of *Geolog* trees. Theorem 2 is proved in Section 4 using the concept of complete *Geolog* trees. The references [4], [7], [8], provide additional theoretical background.

3. Geolog Trees

The splitting of tapes during Skolem machine operations suggests that tree structures can provide an alternate description. This was depicted in Figure 7 and the concept is quite simple.

Suppose that we are given a *Geolog* theory G . There is only one *Geolog tree* with one node, and that is the tree *true*. This singular tree corresponds to the initial tape of a Skolem machine for G .

Assume that we have a correspondence between Skolem machine tape configurations and *Geolog* trees up to some number k of rule applications. If the Skolem machine would have had b tapes then the *Geolog* tree T_k has a total of b branches. Let us examine a $(k + 1)$ st rule application. This application would have been applied to a particular tape of the Skolem machine. For the *Geolog* tree, the application is at the leaf of the corresponding branch B of the tree.

Consider again the general form of a *Geolog* rule (1). When such a rule (r_1 say) is applied to the current tape it splits into n tapes. The corresponding *Geolog* tree branches instead, as visualized in the following diagram. The leftmost branch (B_1 in the diagram) is an extension defined using a Skolem machine operation corresponding to the first disjunct of the consequent of the rule r_1 . The other new branches (if any) are similarly formed.

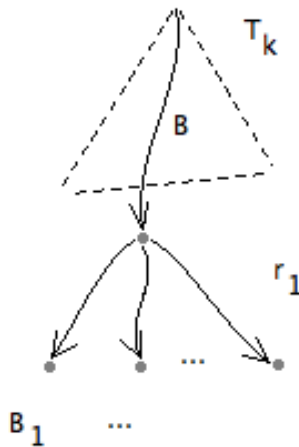


Figure 8. Branching

By induction, any Skolem machine computation (sequence of operations) can be expressed by a corresponding *Geolog* tree.

Suppose that B is a branch (from root to leaf) in a *Geolog* tree. A *branch conjunction* is any conjunction b_1, b_2, \dots, b_i of logical terms which appear at the nodes of the tree on branch B . Suppose that s_1, s_2, \dots, s_j are the distinct eigenvariables appearing in the branch conjunction. Let b'_1, b'_2, \dots, b'_i be the branch conjunction expression with the eigenvariables replaced by distinct variables x_1, x_2, \dots, x_j and then form the logical formula $(\exists x_1, x_2, \dots, x_j)(b'_1, b'_2, \dots, b'_i)$. A *branch wff* is any such well-formed logical formula, where the ordering of the logical variables is arbitrary and the ordering of the conjuncts is also arbitrary.

A *tree wff* is any disjunction $c_1 \mid c_2 \mid \dots \mid c_p$ such that for any branch B of the *Geolog* tree, one of the c_j is a branch wff for B and each c_j is a branch wff of the tree.

For example, $(\exists x)p(x)$ is a tree wff for the *Geolog* tree in Figure 7, and so is $(\exists x)(q(x), u(x)) \mid (\exists y)u(y) \mid (\exists x, y)s(x, y)$.

Proposition. *If w is a tree wff for a tree based on a Geolog theory then w is a logical consequence of the axioms of the theory.*

Proof. The proposition is vacuously true in the case that there are 0 rule applications to build the tree; this is just the tree *true*. Suppose that the proposition is true whenever k rule applications build the tree. Assume that $k + 1$ rule applications built our tree. The last rule application can again be depicted as in Figure 8. Consider a tree wff w for this tree. We can express w as $w = w_k \mid w_r$ where w_r is the disjunction of branch wffs from B expanded using rule $r = r_1$, and w_k consists of the other branch wffs. If any branch wff in w_r is formed using only facts along B proper, then $w = w_k \mid w_r$ is a logical consequence of the axioms, by the induction hypothesis (ignore the application of r). Otherwise, let us write the instance of the expanding rule r as

$$a_1, a_2, \dots, a_n \Rightarrow c_1 \mid c_2 \mid \dots \mid c_m \quad (7)$$

Here the facts a_1, \dots, a_n occur along B . We can also now express w_r as $w_r = w_1 \mid \dots \mid w_m$ where branch wff w_i contains at least one conjunct formed using c_i ($i = 1, \dots, m$).

Let b be the conjunction of all the facts on branch B , among which are a_1, \dots, a_n and consider $v'_i = \exists(b', c'_i)$ where the existential quantifier captures all of the eigenvariables (if any), $i = 1, \dots, m$. Now each w_i is a logical consequence of the corresponding v_i . Moreover, $v_1 \mid \dots \mid v_m$ is a logical consequence of $\exists b'$ where the latter closes b with existential quantification. And so $w_r = w_1 \mid \dots \mid w_m$ is a logical consequence of $\exists b'$ and the axioms of the theory. Now this makes $w = w_k \mid w_r$ a logical consequence of $w_k \mid \exists b'$ and the axioms of the theory. But $w_k \mid \exists b'$ is itself a logical consequence of the axioms, by the induction hypothesis, because $w_k \mid \exists b'$ is a tree wff formed inside T_k . Therefore, $w = w_k \mid w_r$ is also a logical consequence of the axioms of the theory, as required. \square

Notice that the set of all facts along any branch of the tree is a model for a tree wff. Call these models *branch models* of the tree wff.

Proof of Theorem 1. Suppose that the theory supports its query Q . Consider the tree corresponding to the halted Skolem machine. A subdisjunction Q' of Q is the tree wff for this tree. According to the Proposition, Q' , and hence Q , is a logical consequence of the axioms of the theory. \square

It is worth noting, in regards to the Proposition, that well-formed formulas constructed by (for example) existentially quantifying eigenvariables *after* disjoining branch facts, $\exists(- \mid - \mid \dots \mid -)$, are

also logical consequences of the axioms of the theory. These consequences constitute possible *answers* to the query. These wffs may indeed be stronger consequences, but they do not have the geometric (or coherent) form that queries have.

The reference [8] defines *complete Geolog trees* and uses them to prove Theorem 2. Roughly speaking, complete trees require that all applicable rules be used to expand trees in stages, and this is the topic of the next section.

4. Complete Geolog Trees

To motivate the general definitions, consider first the following simple Geolog theory, G_1 .

```

true => a | b .    % #1
true => c, d .    % #2
a => goal .       % #3
b, c => e .       % #4
e, d => false .   % #5
    
```

For the definition of a *Geolog trees* we consider the Geolog theory itself to be an *ordered* sequence of Geolog rules. Reference will be made to the rules of theory G_1 using their serial order (display notation: #n). The order will turn out to be irrelevant to the *branch sets* defined by the branches in these trees, and the branch sets will be the important semantic objects: They will be partial logical models (or possibly counter-models).

A complete Geolog tree of level 0, for any ordered Geolog theory, consists of just the root node *true*. The level 0 tree is, obviously, independent of the rule order. Figure 9 shows the complete Geolog tree of level 1 for the ordered theory G_1 .

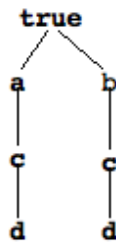


Figure 9. Complete tree for G_1 , level 1

The root of any Geolog tree is the unique atom *true*, which is the complete Geolog tree of level 0. The complete level 1 tree expands (and extends) the level 0 tree.

The first applicable rule for level 1 in our example is #1, and this constructs two branches for the growing tree. The second applicable rule (#2) adds elements to the growing tree along both branches because *true* is an ancestor for both branches. Notice that the consequents maintain a similar order of appearance (specifically, top-down) in the tree, as they appear in the consequence of rule #2 (specifically, left-to-right).

At level 2, rule #3 applies to the left branch of the complete tree for level 1, and rule #4 applies to the right branch, so a graphical depiction of the complete Geolog tree for G_1 for level 2 is given in Figure 10.

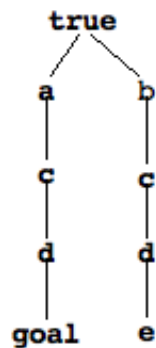


Figure 10. Complete tree for G_1 , level 2

Finally, at level 3, rule #5 applies to the right branch in Figure 10, as shown in Figure 11. At this stage, level #3, the tree is *saturated* because each branch contains either *goal* or *false*.

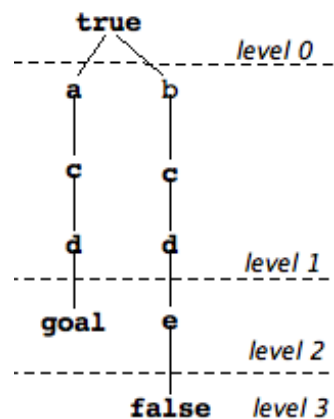


Figure 11. Complete tree for G_1 , level 3, with levels marked

Now let us suppose that the rules in the theory G_1 are reordered, for example

```

true => c, d .    % #2
true => a | b .  % #1
b, c => e .      % #4
e, d => false .  % #5
a => goal .      % #3
  
```

In this case the complete Geolog trees of levels 0, 1, 2, and 3 could be depicted as shown in Figure 12.

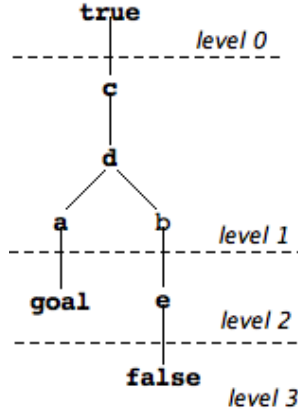


Figure 12. Complete tree rules reordered, level 3, with levels marked

Notice that the level branch sets are the same. A *branch set* for level k consists of the set of all facts on a branch of the complete level k tree from the root of the tree down to the leaf of the branch. The branch sets for either tree, Figure 11 or 12 are

- level 0: {true}
- level 1: {a,c,d}, {b,c,d}
- level 2: {a,c,d,goal}, {b,c,d,e}
- level 3: {a,c,d,goal}, {b,c,d,e,false}

The query for theory G_1 is $Q = a \mid d, e$ and the level 3 branch sets also represent successfully halted tapes for a Skolem machine for G_1 .

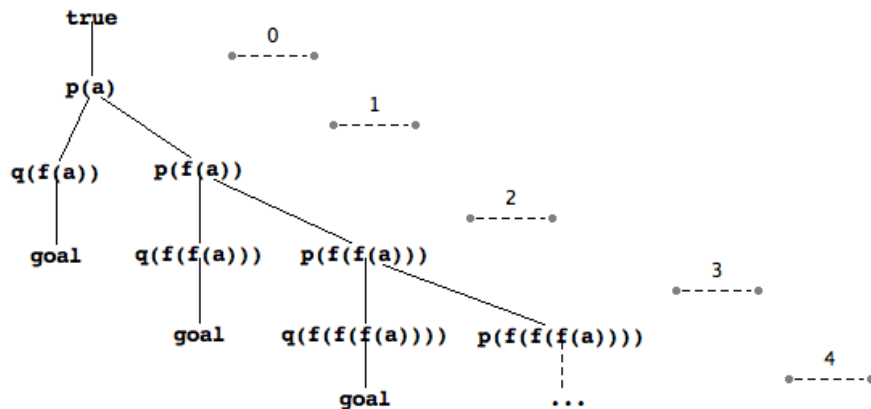
Another example is afforded by the following Geolog theory, G_2 . G_2 does not support its query. Figure 13 shows some of the complete trees for G_2 .

```

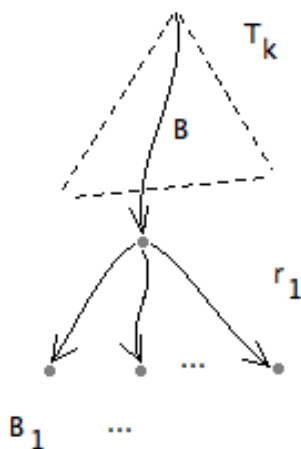
true => p(a) . % #1
p(X) => q(f(X)) | p(f(X)) . % #2
q(X) => goal . % #3
  
```

For G_2 the complete trees are unbounded, meaning simply that the number of nodes in the tree grows without bound as the level increases. A corresponding Skolem machine would have an unbounded number of possible tapes.

More formally, suppose that G is an arbitrary Geolog theory. We define a complete Geolog tree for G of level k by induction on k . The unique complete Geolog tree T_0 of level 0 for G is just the root tree, already describe. The single branch set for T_0 is $\{true\}$. Suppose that T_k is the complete Geolog tree for G of level k having branch sets B_i . It is assumed that any branch of T_k which contains either *goal* or *false* has that node as a leaf of the branch. Then T_{k+1} is defined as follows. The branches having leaf *goal* or *false* are not extended; they are considered to be *saturated*. For any branch B of T_k not having leaf *goal* nor *false* let us assume that r_1, r_2, \dots, r_z is a complete ordered list of all possible applicable instances of *geolog* rules which are not already satisfied on B . We assume that the specific order is determined by the order that the rules are given in G . These ground instances may have arisen from

Figure 13. The infinite tree for $G_2 \dots$

the same or from different rules, but there are only *finitely many* such instances. Use r_1 to extend B in the same way as if B were a corresponding Skolem machine tape, as described in the previous section. However, if r_1 is a splitting rule, then split the branch B of T_k rather than reproduce the tape B and then extend the copies. (If r_1 is not a splitting rule then B has a unique extension.) Assume that this produces m branches B_1, \dots, B_m , as shown in Figure 14. (Figure 8 uses the same graphic, but the tree in Figure 8 was not required to be complete. In the earlier figure, k represents the k th rule application, and in the current figure, k represents a “level” for possibly many rule applications.)

Figure 14. Expanding branch B of complete tree using first applicable rule

If any of the extended branches B_j has leaf *goal* of *false*, that branch is considered to be saturated, and it is not extended (or expanded) any further. Continuing, we now apply r_2 to each of the new branches not having leaf *goal* of *false*, then r_3 to the resulting branches, until all of the rules $\dots r_z$ have been used to expand all of the previous branches not having leaf *goal* of *false*, using the process described for r_1 ,

corresponding to Figure 8. The tree T_{k+1} is the result of this double induction for all branches B of T_k and all resulting applicable rules for each B (but never expand leaf *goal* of *false*).

Theorem 2. Suppose that Q is the query for Geolog theory G and that Q is a logical consequence of G . Then G supports Q .

Proof Sketch. The collection of all complete Geolog trees T_k for $k = 0, 1, 2, \dots$ defines a (possibly infinite) tree T . Each node in T has finitely many children. Branch sets correspond to Herbrand models (closed term models) in the usual sense [9], but with the Herbrand basis based on the signature *plus the generated constants*. Note that, by construction, for each branch set B of T_k , any false instance of any rule is applicable and hence satisfied in all extensions of B in T_{k+1} .

If T is a finite tree and some branch set B does not satisfy any of the disjuncts of Q then B would satisfy the axioms of G but not Q . Since Q is a logical consequence of the axioms of G this case is not possible and so if T is finite then the branch sets of T correspond to a successfully halting Skolem machine and so G supports Q .

If T is infinite then, by König's lemma [11], T has an infinite branch. If none of the branch sets corresponding to this infinite branch satisfies any disjunct of Q then the set of nodes on this branch is a counter model. Since Q is a logical consequence of the axioms of G this case is not possible. Thus T is in fact a finite tree, and every disjunct of Q is satisfied on one of the branch sets of T . \square

5. A Universal Skolem Machine

A machine U is *universal* for a class \mathcal{C} of machines if for every $M \in \mathcal{C}$ and every input I for M , when given input $\lceil M \rceil, \lceil I \rceil$, it mimicks the behaviour of M when given input I . Here ‘mimicks’ means in particular that U with input $\lceil M \rceil, \lceil I \rceil$ terminates if and only if M with input I does so. Moreover, it is required that if M with input I returns output O , then U with input $\lceil M \rceil, \lceil I \rceil$ returns output $\lceil O \rceil$. (We use $\lceil M \rceil, \lceil I \rceil, \lceil O \rceil$ to stress the difference between M, I, O and their representation in the format that U uses.)

Of course the most famous universal machine is the one for the class of Turing machines, used by Turing to prove the undecidability of the halting problem. If the class \mathcal{C} of machines is Turing-complete, then the halting problem for any machine that is universal for \mathcal{C} is undecidable. Therefore the construction of a universal machine is still important, theoretically as well as in practice, where the concept of a universal machine plays a role as a so-called *interpreter*.

In this section we will construct a universal Skolem machine (USM). It would be possible, analogous to what Turing did, to construct a Skolem machine which is universal for the class of all Skolem machines, but this would involve very many technical details. In order to minimize the amount of detail we take as class \mathcal{C} the class of so-called 2-counter machines which is known to be Turing-complete [13]. Using a USM (designed below) for \mathcal{C} one immediately infers that Skolem machines have the same computational power as Turing machines and that, consequently, even tiny fragments of geometric logic are undecidable.

A counter machine is a device with *counters* x_1, \dots, x_m , each capable of storing an arbitrarily large natural number, together with a program. A counter machine program is a finite enumeration of *instructions* from the following instruction set: $inc(x_i), dec(x_i), jpz(x_i, l, l')$. These instructions lead to the

following respective actions: *increment* counter x_i , *decrement* counter x_i , *jump* to instruction l if x_i is zero and to l' otherwise. Decrementing a counter which has value 0 is not allowed and can be prevented by using conditional jumps preceding any decrement instruction.

The execution model for counter machines uses one additional counter, the so-called *program counter*, which references the current instruction. Execution of the program starts at the first instruction. The program counter is incremented after each instruction $inc(x_i)$, $dec(x_i)$, its value is changed to either l or l' in case of a conditional jump. Execution terminates when the program counter gets a value not corresponding to an instruction of the program.

In [13], Minsky proved that this simple machine model is already Turing-complete when only two counters are used. As an example consider the following program:

```

1   $jpz(x_2, 5, 2)$ 
2   $dec(x_2)$ 
3   $inc(x_1)$ 
4   $jpz(x_1, 1, 1)$ 

```

This program obviously adds the contents of x_2 to x_1 and terminates by jumping to 5, beyond the last instruction. Instruction 4 exhibits an unconditional jump. (The program would also work correctly with instructions $4\ jpz(x_{13}, 1, 1)$ or $4\ jpz(x_2, 13, 2)$ instead of $4\ jpz(x_1, 1, 1)$.)

A *Geolog* theory that is universal for any 2-counter program is displayed in Figure 15. It is important to observe that there are *no function symbols* in the theory so Skolem Machines are universal without function symbols (unlike Datalog programs which are not universal, but are universal if one is allowed the use of but a single function symbol.) Intuitively the justification for this is that *Geolog* allows existential quantification in the consequent of a rule. Notice that there is exactly one existential quantification in the theory of Figure 15, in the very last rule which is used to generate the natural numbers!

6. Implementation Design

In definite logic programming theory, *SLD deduction* provides top-down derivations of top-level goals. (Select goal - use Linear resolution - for Definite rules). See [12] for concise characterization of SLD deduction for definite logic programs.

For Skolem machine operations on a *Geolog* theory, the STG acronym has meanings reminiscent of those SLD deduction for definite logic, but with significant modifications.

STG deductions for Geometric logic theories provide bottom-up derivations from bottom-level facts, but the STG operations build *Geolog* trees top-down, using the operations of a Skolem machine.

- S - Select an applicable rule instance (A single rule can have multiple applicable instances.)
- T - use facts on the current branch of the *Geolog* tree and extend until saturation
- G - using geometric logic rules

STG deductions grow *Geolog* trees as described in Section 3. Facts on the current branch are used to create applicable instances of rules. The first disjunct of the consequent of the selected rule is used to extend the tree using the leftmost disjunct, and save the other disjuncts for subsequent branching only after the current branch becomes saturated. If the current branch is saturated and has a *goal* or *false* leaf, then branch the tree at the deepest remaining branch point above the leaf, continuing to grow the new branch using the next disjunct not previously used.

Figures 16-19 illustrates a step-by-step STG deduction for the sample theory of Figure 1. The steps in the STG deduction correspond in a natural way to the steps used in the Skolem machine operations in Section 2. The facts in the trees with the red dashed outline indicate the current *focus* of the deduction.

Figures 17(f) and 18(j) illustrate later branching after completing a branch corresponding to a previous disjunct in the consequent of a rule. The focus is an internal (non-leaf) node in these cases.

Provided that we can use arbitrary rule application instances, STG deduction is correct and complete.

Theorem 3. Suppose that Geolog theory G supports its query Q with tree T. Then there is an STG deduction which builds T.

Proof. A inorder traversal of T will construct a selection function for which particular rule applications to make in order to reproduce the same tree with an STG deduction. □

We do not expand upon the topic in this paper, but an important concept is the *extraction of answers* from a successful STG deduction.

There is, of course, no effective algorithm for computing effective rule instance selection functions in general.

One selection strategy that is easy to implement is the *first applicable rule instance selection function*: Find the first applicable rule, in their given order, whose antecedent can be matched against facts on the current branch of the tree, from the top down, and whose consequent is not already satisfied using the matching bindings for the antecedent.

Implementations of STG with the first selection function, using Prolog procedures can be very straightforward. One can rely of Prolog's unification for matching facts, and Prolog's backtracking for scheduling rule applications.

Each of the *Geolog* rules in the instruction set is translated into a special kind of Prolog clause. The implementation that we illustrate is called the *Skolem Abstract Machine* or SAM for short. The reason for this name is that the clauses resemble the procedures of the *Warren Abstract Machine* (WAM), which is used as a basis for most of the efficient implementations of Prolog itself. In particular, each procedure tries to match bindings for variables in terms. For the SAM procedures, however, the terms can be in different states (multiple tapes, different tree branches).

The outline given in the remainder of this section can serve as motivation for lower-level implementations of the SAM, and for modifications to the selection function.

Some cogent references for the machine model and implementation of the WAM are [1] and [18].

The Prolog translator is basically a one-line program that mimics the STG operations, using the first selection function. The `translate` rule has the profile shown in Figure 20.

Figure 21 shows the full Prolog code for translating a *Geolog* rule.

For example, consider *Geolog* rule #2 from the sample theory in Figure 1, repeated in Figure 22. First, any existential variables in the consequent are separated and flagged, as shown in Figure 23, and then the translated Prolog clause is displayed in Figure 24.

The code for the translator in Figure 21 mimics the definition of how STG deduction applies would use a first applicable rule instances to extend tree branches. The Prolog code implements the tree branch using a Prolog list, and terms are added to the beginning of the list (end of the branch). (Faster implementations use *memoing* rather than lists, but the code presented here may be easier to understand.)

Each of the `try` clauses describes how to *try* to extend a tape using the corresponding *Geolog* rule. The *Geolog* rules are translated into Prolog clauses in the order in which they appear in the *Geolog* instruction sequence. Figure 25 has an outline for all of the `try` clauses, showing the order in which they are asserted to memory (and compiled).

In the SWI-Prolog [19] implementation of the SAM, after the *Geolog* rules are read from file and translated into Prolog clauses, the Prolog clauses are asserted and then compiled into internal procedures for the underlying Prolog machine.

The remaining small amount of code for applying *Geolog* rules, expanding, and extending tapes (states) is given in the reference [6]. The Prolog interpreter has filename `geoprolog.pl`. The reference also provides a user guide and numerous sample *Geolog* theories to compute.

As it is for Prolog, the ordering of *Geolog* instructions becomes important for the first applicable rule selection function implementation of SAM.

As emphasized in [3], it is often best to sequence the instructions so that splitting rules and rules introducing existential quantifiers are placed *at the end* of the rulebase, such as the rules 12 and 13 in Figure 26. Moving these rules higher up in the list is inhibitive for computing the query depth-first. Using the first selection function and given ordering one produces the proof displayed in Figure 27. (An interesting detail is that rule 9 is not necessary for proof, but deleting it makes for a longer proof, 111 steps.)

In the presence of function symbols even more trivial examples can be given, such as the wrong order of the last two of the following rules:

```

true => p(a).
p(X) => p(f(X)). %alternatively: p(X) => succ(X,Y),p(Y).
p(X) => goal.

```

7. Conflicted Geolog Theories

Recall that we can write a *Geolog* theory in the form

$$T = A \cup G \cup F \tag{8}$$

where A is the *axioms*, G contains all of the *goal* rules and F contains all of the *false* rules.

Also recall that the *geolog query* Q for a *Geolog* theory $T = A \cup G \cup F$ is the disjunctive normal form $Q = C_1 \mid C_2 \mid \dots \mid C_k$ where the C_i are the existential closures of the antecedents of all the rules in G and F . A *Geolog* theory T *supports* its query Q provided there is a successfully halted Skolem logic machine such that each tape of the halted machine satisfies at least one of the C_i .

Notice that any theory lacking both *goal* and *false* rules cannot support its empty query, since there is no way for a Skolem machine to successfully halt.

We say that a theory $T = A \cup G \cup F$ is *conflicted* provided that $G \neq \emptyset$, T supports its query, and the theory $T - G = A \cup F$ also supports its query. If $T - G$ supports its query, then necessarily $F \neq \emptyset$.

Observation. If $T = A \cup G \cup F$ is conflicted then T 's query is not a minimal logical consequence of A .

To say that $Q = C_1 \mid C_2 \mid \dots \mid C_k$ is not a minimal logical consequence of A means that there is a properly smaller disjunction that is also a logical consequence. A simple example would be the theory T :

```

true => a,b.    % A
a => goal.      % G
b => false.     % F

```

Here we have that T 's query $Q = a \mid b$, and both a and b are logical consequences of A . Notice that $T - G$ also supports its query $Q' = b$, and so Q is not a minimal consequence of the axioms of T .

The converse of the theorem is not true. For example, consider the previous theory with `false` replaced by `goal`. $Q = a \mid b$ is not a minimal consequence but $T - G$ has no *goal* or *false* rules, so cannot support its empty query. That is, both a and b are logical consequences of the axioms, so $a \mid b$ is not a minimal logical consequence, but the theory is not conflicted. So a and b would be better *answers* as logical consequences of the theory.

By convention, *goal* is often thought of as "affirming" and *false* as "rejecting", so the definition of conflicted favors *goal* as the "positive" concept.

The theory given in Figure 26 is an example where *goal* affirms and *false* rejects. This theory supports its query $Q = e(b, c) \mid (\exists z)r(b, z) \mid (\exists z)r(c, z)$ but $T - G$ does not support its query $Q' = (\exists z)r(b, z) \mid (\exists z)r(c, z)$. Thus, the theory is not conflicted. The programmer's intention was to show that

$$\neg(\exists z)r(b, z) \wedge \neg(\exists z)r(c, z) \rightarrow e(b, c) \quad (9)$$

and to disprove the negation of the antecedent:

$$(\exists z)r(b, z) \vee (\exists z)r(c, z) \quad (10)$$

This is indeed the case. Figure 27 establishes (9). It can be shown additionally that $T - G$ does not support its query (10), and that $T - G$ has a finite counter-model. There is, in fact, a Skolem machine computation for $T - G$ with saturated tape not containing *false* which was automatically verified by the implementation similar to the one described in Section 6). Figure 28 illustrates this by displaying the rest of the facts in a counter-model, when the *goal* rule is removed from the theory. Explicitly stated, the counter-model consists of all the facts from 0 to 24 on the left-most branch of Figure 27 together with the saturating facts 25 through 36 shown in Figure 28.

8. Conclusions

This main intent of this paper was to supply essential definitions and theorems for the concept of a Skolem machine, and to explain some of the historical connections for the concept. For this we have defined a simple logical input language, *Geolog*, for Skolem machines, and provided the basic operations

of the logical machine using the *Geolog* rules or instructions. A modest definition of a query, defined solely in terms of the input theory, has been provided, and essential theorems covering correctness and completeness of Skolem machine operations (with regard to computing the query) are proved in some detail. We show how logical correctness follows from a detailed analysis of what we call *Geolog trees* and how logical completeness follows from a detailed analysis of *complete Geolog trees*. This particular tree analysis has promise for further study regarding answer extraction and proof strategies. Stronger logical results (not just for the restricted query) have either been stated (as in the case of correctness) or outlined for further development.

The universality of Skolem machine calculations has been established using arguments based upon the direct simulation of a *universal 2-counter register machine*. The arguments established the interesting result that no function symbols are need for universality.

On the implementation side, we have provided a depth-first deduction procedure, called STG deduction, which is pseudo-complete (up to selection or rule choice function), and we have described a simple, but effective translation into Prolog using the so called *first choice* selection function, or strategy.

And finally, the paper provides a justification for using, or allowing, two kinds of terminal conclusions for *Geolog* rules: *goal* or *false*, and explains that the distinction is merely a handy formalism that is useful for extending the meaning of Skolem machine computations.

The next steps in this work involve more efficient and effective implementation of theorem provers or model checkers whose underlying computational mechanism is the Skolem machine. This may eventually involve extensions to the *Geolog* language.

Much more might be said regarding historical connections between what we call *Skolem Machines* and the work of Thoralf Skolem. These connections are not pursued here. A good perspective regarding Skolem's influence on logic is the reference [14].

References

- [1] H. Ait-Kaci, *Warrens's Abstract Machine, A Tutorial Reconstruction*, School of Computing Science, February 18, 1999.
- [2] M. Bezem and T. Coquand. Newman's Lemma – a Case Study in Proof Automation and Geometric Logic. In Y. Gurevich, editor, *The Logic in Computer Science Column, Bulletin of the European Association for Theoretical Computer Science* **79**:86–100, February 2003. Also in G. Paun, G. Rozenberg and A. Salomaa, editors, *Current trends in Theoretical Computer Science*, Volume 2, pp. 267–282, World Scientific, Singapore, 2004.
- [3] M.A. Bezem and T. Coquand, Automating Coherent Logic. In G. Sutcliffe and A. Voronkov, editors, *Proceedings LPAR-12*, LNCS 3835, pages 246–260, Springer-Verlag, Berlin, 2005.
- [4] Marc Bezem. On the Undecidability of Coherent Logic. In Aart Middeldorp e.a., editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, LNCS 3838, pages 6–13, Springer-Verlag, Berlin, 2005.
- [5] A. Blass, Topoi and computation. *Bulletin of the EATCS* **36**:57–65, October 1998.
- [6] *Geolog* website: <http://www.csupomona.edu/~jrfisher/www/geolog>
- [7] John Fisher and Marc Bezem, Skolem Machines and Geometric Logic. In C.B. Jones, Z. Liu and J. Woodcock, *Proc. ICTAC 2007 The 4th International Colloquium on Theoretical Aspects of Computing*, Macao SAR, China, September 26-28, 2007. Springer LNCS vol. 4711, pp. 201-215.

- [8] John Fisher and Marc Bezem, Query Completeness of Skolem Machine Computations. In J. Durand-Losé and M. Margenstern, editors, *Proc. Machines, Computations and Universality '07*, Université d'Orléans - LIFO, Orleans, France September 10-14, 2007. Springer LNCS vol. 4664, pp. 182-192.
- [9] Jacques Herbrand, *Logical Writings*, edited by Warren D. Goldfarb, D. Reidel Publishing Company, Springer edition, 2006.
- [10] P. Johnstone, *Sketches of an Elephant: a topos theory compendium*, Volume 2, Oxford Logic Guides 44, OUP, 2002.
- [11] Dénes König, *Theorie der endlichen und unendlichen Graphen*, Akademische Verlagsgesellschaft, Leipzig, 1936. Translated from German by Richard McCoart, *Theory of finite and infinite graphs*, Birkhauser, 1990.
- [12] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Berlin, revised edition 1987.
- [13] Marvin L. Minsky, Recursive unsolvability of Post's problem of 'tag' and other topics in theory of Turing machines, *Annals of Mathematics* **74**(3):437–455, 1961.
- [14] *Nordic Journal of philosophical Logic* **1**(2) December 1996. Special issue devoted to the influence on logic of Thoralf Skolem.
- [15] R. Manthey and F. Bry, SATCHMO: A Theorem Prover Implemented in Prolog. In: E. Lusk and R. Overbeek, editors, *Proceedings CADE-9*, LNCS 310, pages 415-434, Springer-Verlag, Berlin, 1988.
- [16] H. de Nivelle and J. Meng, Geometric Resolution: A Proof Procedure Based on Finite Model Search. In: U. Furbach and N. Shankar, editors, *Automated Reasoning*, Proceedings IJCAR 2006, LNCS 4130, pages 303–317, Springer-Verlag, Berlin, 2006.
- [17] Thoralf Skolem, *Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit und Beweisbarkeit mathematischen Sätze nebst einem Theoreme über dichte Mengen*, Skrifter I **4**:1–36, Det Norske Videnskaps-Akademi, 1920. Also in Jens Erik Fenstad, editor, *Selected Works in Logic by Th. Skolem*, pp. 103–136, Universitetsforlaget, Oslo, 1970.
- [18] D.H.D. Warren, *Implementation of Prolog*. Lecture notes of Tutorial No. 3, 5th International Conference and Symposium on Logic Programming, Seattle, WA, August 1988.
- [19] J. Wielemaker, *SWI-Prolog Reference Manual*. Link available at: www.swi-prolog.org

```

%% 2-Counter machine
%%   Example: R1 + R2 -> R1
%%   1: jpz(R2,5,2)
%%   2: dec(R2)
%%   3: inc(R1)
%%   4: goto(1)
%%   5: halt
%%
true =>
  % program
  instruction(1,jpz,2,5,2),
  instruction(2,dec,2,3,x),
  instruction(3,inc,1,4,x),
  instruction(4,goto,1,x,x),
  instruction(5,halt,x,x,x),
  % data
  inc(0,1), inc(1,2), inc(2,3), inc(3,4),
  state(1,3,4). // START compute 3 + 4

%% UNIVERSAL INTERPRETATION OF REGISTER INSTRUCTIONS
%-- jpz
state(PC,0,R2), instruction(PC,jpz,1,PA,PB) => state(PA,0,R2).
state(PC,R1,R2), inc(_X,R1), % R1 is NOT zero
  instruction(PC,jpz,1,PA,PB) => state(PB,R1,R2).
state(PC,R1,0), instruction(PC,jpz,2,PA,PB) => state(PA,R1,0).
state(PC,R1,R2), inc(_R2), % R2 is NOT zero
  instruction(PC,jpz,2,PA,PB) => state(PB,R1,R2).
%-- dec
state(PC,R1,R2), instruction(PC,dec,1,PB,x), inc(D,R1) => state(PB,D,R2).
state(PC,R1,R2), instruction(PC,dec,2,PB,x), inc(D,R2) => state(PB,R1,D).
%-- inc
state(PC,R1,R2), instruction(PC,inc,1,PB,x), inc(R1,I) => state(PB,I,R2).
state(PC,R1,R2), instruction(PC,inc,2,PB,x), inc(R2,I) => state(PB,R1,I).
%-- goto
state(PC,R1,R2), instruction(PC,goto,PB,x,x) => state(PB,R1,R2).
%-- halt
state(PC,R1,R2), instruction(PC,halt,x,x,x) => goal.

%% Natural number generation via inc
inc(X,Y) => inc(Y,SomeZ).

```

Figure 15. A Universal Skolem Machine

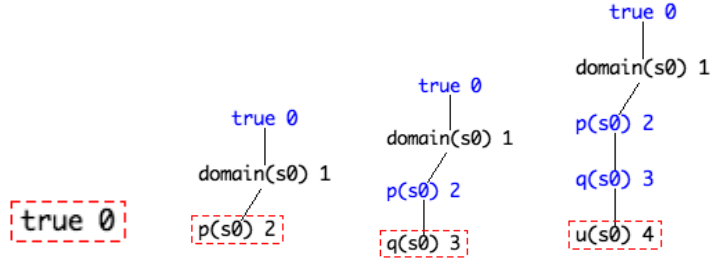


Figure 16. STG (a,b,c,d)

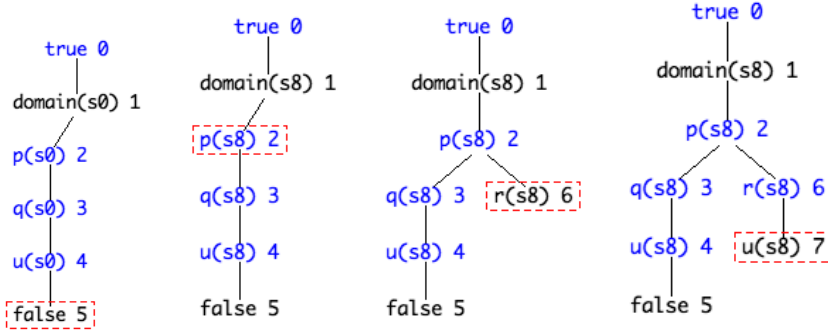


Figure 17. STG (e,f,g,h)

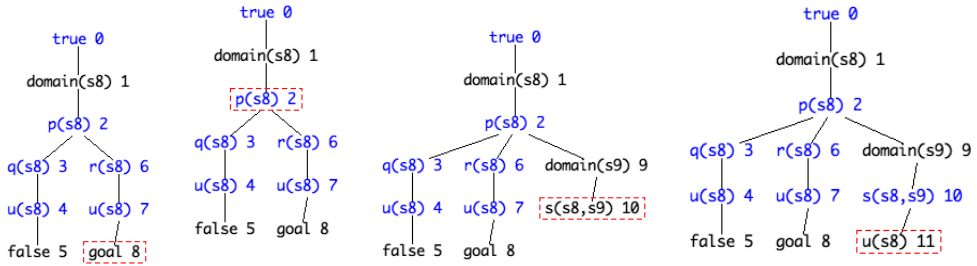


Figure 18. STG (i,j,k,l)

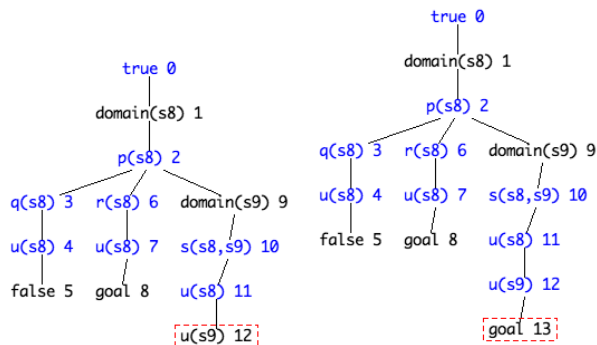


Figure 19. STG (m,n)

```
translate(+GeologRuleIn, -PrologRuleOut)
```

Figure 20. The intended translation, *Geolog* rule to Prolog clause

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Geolog Translator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
translate((ANT => CONS) , % to the following Prolog clause ...
  (try((ANT => CONS),Branch) :-
    satisfy(ANT,Branch),
    \+satisfied(CONS,Branch),
    cases(CONS,[F|R]),
    extend(F, Branch,FBranch),
    try(_,FBranch),          % try again
    continue(Branch,R) ) ) . % other cases, if any
```

Figure 21. Translating *Geolog* rules to Prolog clauses

```
p(X) => q(X) | r(X) | domain(Y), s(X,Y).
```

Figure 22. +GeologRuleIn, sample input term

```
p(X) => q(X) | r(X) | Y^(domain(Y), s(X,Y)).
```

Figure 23. +GeologRuleIn, flag existential variable

```

try((p(A)=>q(A)|r(A)|B^ (domain(B), s(A, B))), C) :-
  satisfy(p(A), C),
  \+satisfied((q(A)|r(A)|B^ (domain(B), s(A, B))), C),
  cases((q(A)|r(A)|B^ (domain(B), s(A, B))), [E|H]),
  extend(E, C, F),
  try(_, F),
  continue(C, H).

```

Figure 24. -PrologRuleOut, sample output term

```

% START with initial state
try :- try(_, [true]).

% test for goal on tape (or \f )
try(_, S) :-
  member(goal, S), member(false, S), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Translated Geolog clauses asserted
%% here, in user-specified order.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% last clause, must have stuck tape
try(_, S) :-
  write('counter_model('),
  write(S),
  writeln(').')

```

Figure 25. Order for the translated Prolog clauses

```

true => domain(a), domain(b), domain(c). %1 domain elements a,b,c
e(b,c) => goal. %2 the goal is to prove b=c
r(b,Z) => false. %3 for normal form b
r(c,Z) => false. %4 and normal form c
true => s(a,b),s(a,c). %5 both reducts of a
domain(X) => e(X,X). %6 reflexivity of e
e(X,Y) => e(Y,X). %7 symmetry of e
e(X,Y),e(Y,Z) => e(X,Z). %8 transitivity of e
e(X,Y),r(Y,Z) => r(X,Z) . %9 r contains e and r
e(X,Y) => s(X,Y). %10 s contains e
r(X,Y) => s(X,Y). %11 and r,
s(X,Y),s(Y,Z) => s(X,Z). %12 is transitive,
s(X,Y),s(X,Z) => domain(U),s(Y,U),s(Z,U). %13 satisfies diamond, and
s(X,Y) => e(X,Y)|domain(Z),r(X,Z),s(Z,Y). %14 is included in e + r.s

```

Figure 26. A *Geolog* theory expressing that confluence of a rewrite relation r implies uniqueness of normal forms

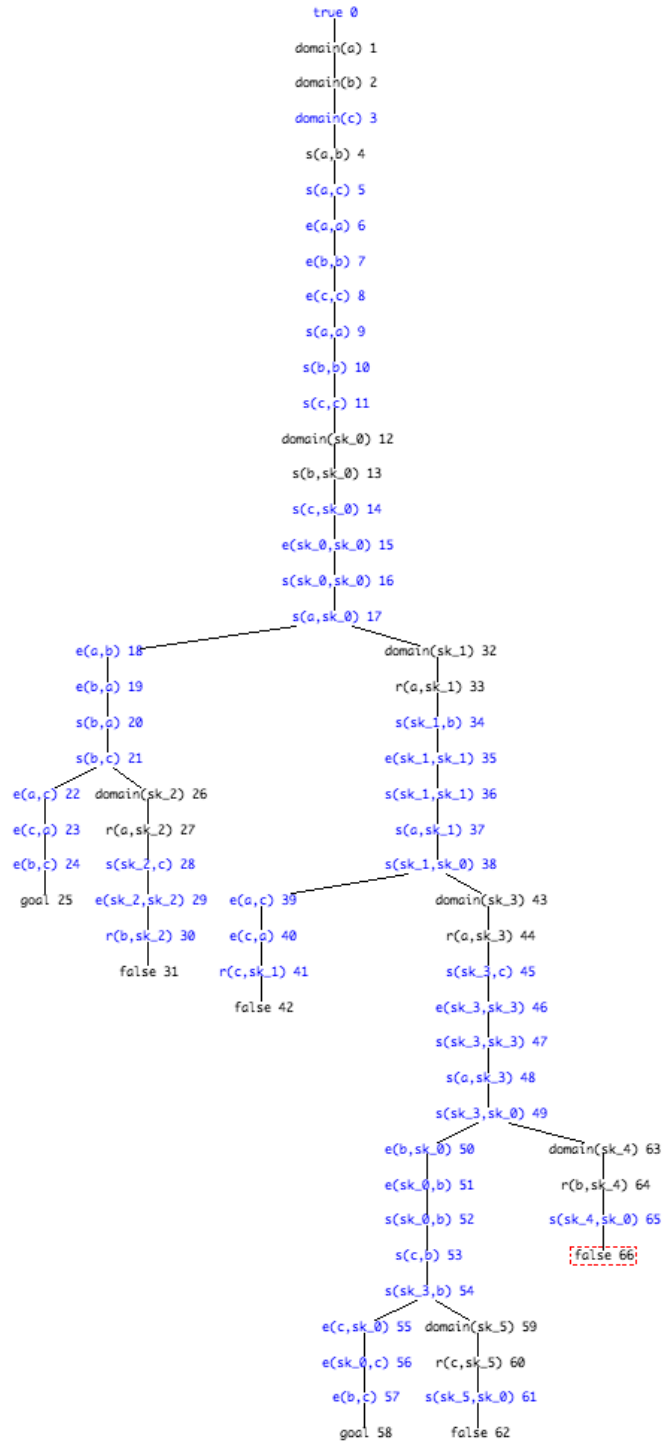


Figure 27. A proof tree for theory in Figure 26

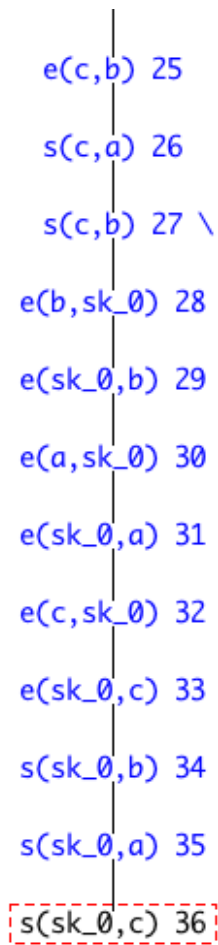


Figure 28. Generating remainder of counter-model for theory of Figure 26 without *goal* rule