

Complete Depth-first Proof Search for Coherent Logic

J.Fisher
Winter 2009

Abstract

This technical note describes an enhanced depth-first algorithm that searches for proofs for coherent logic theories. The coherent theories are expressed as a finite set of geolog rules. The proof search uses geolog trees. The current branch is searched top-down in order to satisfy the antecedent of a geolog rule. The definite rules are allowed to saturate a branch with consequences, but the indefinite existential rules are given turns via queue waiting. In addition, the queued rules use a kind of earliest-first satisfaction regimen, in order to not miss matches with facts and promote the likelihood of termination.

The principle advantage of the *QEDF* algorithm specified is theoretical completeness for coherent theory proof search in the case of theories without function symbols.

After a discussion involving theories without function symbols, we extend the *QEDF* algorithm to theories having function symbols. The extension requires breaking cycles of functional rules within the theory using the queue.

1 QEDF search procedure defined

Suppose that C is a coherent theory specified as a finite ordered sequence r_1, \dots, r_n of geolog rules, as described in [2],[3],[4]. Until section 3, we explicitly assume that the geolog rules do not contain functions in the arguments of the predicates.

The depth-first search algorithm in [2] is modified so that existential rules Q are distinguished from the non-existential, or definite, rules D . The existential geolog rules are those that have at least one free (existential) variable in the consequent of the rule. We have $C = D \cup Q$, as sets, but we assume that the ordered sequences D and Q are relatively ordered as they are in C .

We assume that Q is enhanced with circular queue operations *remove* and *add*. In particular $r = \text{remove}(Q)$ followed by $\text{add}(Q, r)$, would removed r from the front of Q and move it to the rear of the Q queue. The purpose of queueing Q is to limit the generation, or introduction, of new eigenvariables (a.k.a. Skolem constants) in a manageable fashion.

We want to impose the restriction that existential rules are satisfied using *earliest-facts first*. A simple demonstration of this principle is afforded using the following sample theory. Notice that the second rule is existential: Z is an existential variable.

```
true => p(1), p(2), p(3).  
p(X), p(Y) => q(X,Y,Z). // existential rule
```

Figure 1: Sample theory

Consider the two complete geolog trees for the sample theory depicted in Fig. 2.

In the left tree of Fig. 2, the second rule is satisfied in a kind of lexical order, matching the facts at positions 1, 2, and 3 in the tree. In the right tree, the second rule is satisfied using earliest facts first.

Suppose that an existential rule has antecedent A_1, A_2, \dots, A_n . The antecedent factors might match facts on a branch of the geolog tree at various positions. Suppose that $\langle p_1, p_2, \dots, p_n \rangle$ is a positional

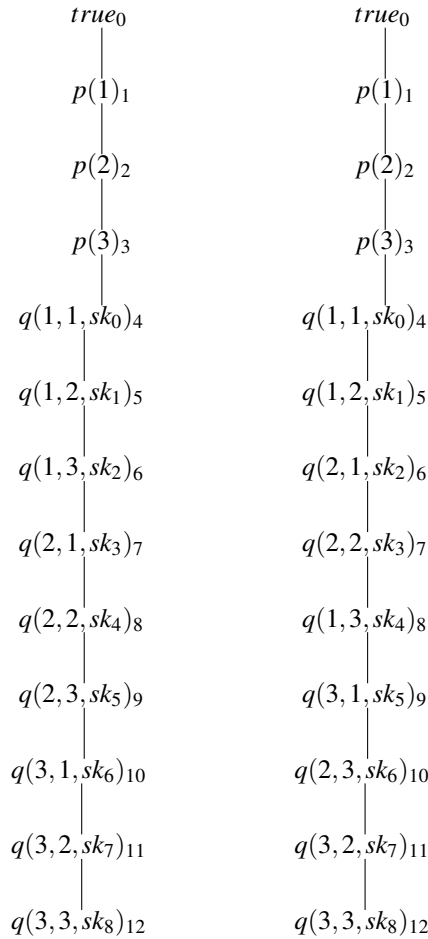


Figure 2: Lexical order (left) vs. earliest-first (right)

match. This means that A_i matches the fact at position p_i in the tree, for $i = 1, 2, \dots, n$. (The matchings must of course be consistent for variable bindings.)

We say that a matching $\langle x_i \rangle$ is *earlier* than matching $\langle y_i \rangle$ provided that $\Sigma x_i < \Sigma y_i$ or else $\Sigma x_i = \Sigma y_i$ but $\max\{x_i\} < \max\{y_i\}$. (Otherwise, the matchings are equally early.) The matchings amount to a kind of *time stamp* for the satisfaction of the antecedent. Roughly speaking, *earlier* means that the average time ($\Sigma x_i/n$) is *higher* on the branch or else the averages are equal but the deepest one is higher in the tree.

The previous theory can be modified slightly in order to illustrate precisely why it is that a lexical order for satisfying the antecedent of an existential rule could avoid matching earlier facts and not produce a proof. The modified theory is listed in Fig. 3.

```

true => p(1), p(2).
p(X), p(Y) => q(X,Y,Z), p(Z). // existential rule
q(2,A,B) => goal.

```

Figure 3: Modified theory

The first rule is modified to make an even simpler example. Notice the change in the second rule. Now, each time the second rule applies it extends the geolog tree by introducing a new fact $p(Z)$ for

a *new witness* Z . If the lexical order of satisfaction illustrated previously is used (where satisfaction is iterated on the second factor of the antecedent), then a depth-first generation of the tree will produce the infinite pursuit depicted in the left tree of Fig. 4, even if one allows all of the definite rules to saturate the branch after each application of the existential rule.

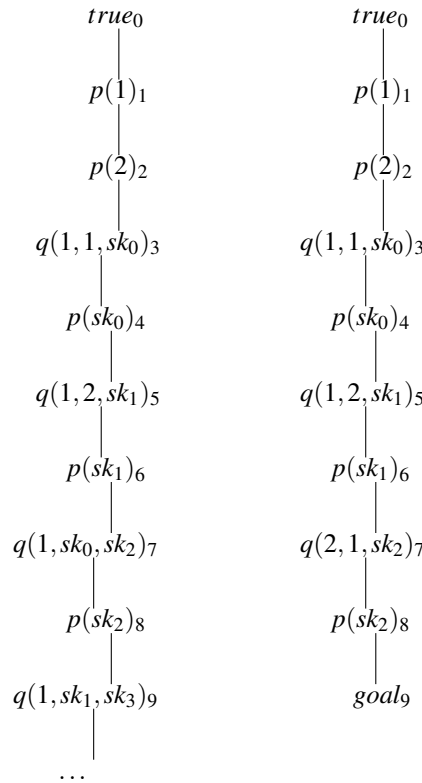


Figure 4: $q(2,-,-)$ unreachable (DF) in left tree, success (QEDF) in right tree

However, if earliest-first satisfaction is used for the existential rule, then we get the simple proof tree depicted on the right in Fig. 4.

Using a different lexical ordering for the satisfaction of factors is not a general solution to the problem. For any ordering one can construct an example where needed inferences could be avoided.

Fig. 5 outlines the basic strategy of the QEDF algorithm. It expresses how definite rules saturate the current branch and existential rules are tried in a queued fashion. In addition, the queued rules are satisfied and applied using earliest-facts-first. The QEDF algorithm is a kind of STG deduction defined in Section 6 of [4]. QEDF has a particular rule selection function and an earliest-fact-first regimen for the queued rules.

Certain details are suppressed in the outline. The algorithm builds a geolog tree in a depth-first fashion starting with the root *true 0*. The antecedents of a rule are match against the facts on the current branch. A rule whose antecedent matches and whose consequent is not already satisfied on the branch can then be applied at the current leaf of the branch. When a rule is applied at the leaf, the branch is extended using the first disjunct of the consequent of the rule, as described in [2], [3] or [4]. Any other disjuncts (along with the index of the branching point) are pushed onto a *choice* stack.

```

// assume tree is initialize with true root
boolean progress = true ;
// D-phase
while (progress) {
    // saturate with D rules
    boolean definite = true ;
    while (definite) {
        definite = false ;
        for (r in D) // original rule order
            if (apply(r)) {
                definite = true ;
                break ;
            }
    }
    progress = false ;
    // all D rules have applied so try some Q rule
    // Q-phase
    for (1 .. Q.size) { // original rule order, queued
        e = Q.remove() ;
        if (applyEF(e)) { // EARLIEST-FIRST satisfaction
            progress= true ;
            break ;
        }
        Q.add(e) ;
    }
}

```

Figure 5: QEDF algorithm outline

If a branch successfully terminates with *goal* or *false*, the choice stack is popped and the next choice is extended at the associated index on the branch, and any remaining choices are again pushed onto the choice stack. The new current branch now extends from the root to the new facts just extended at the branching point index.

Since the definite rules are allowed to saturate the current branch, there is no need for specifying earliest-first satisfaction. But for the queued existential rules, earliest-first satisfaction is assumed in the algorithm. The previous example demonstrates the need for earliest-first satisfaction for the existential rules. In Section 2 we argue that queuing the existential rules is sufficient to make the QEDF algorithm complete for theories without function symbols. In Section 3 we provide more assumptions regarding so that completeness results for all theories..

1.1 avoiding irrelevant branches

To illustrate irrelevant branching, consider the following simple example.

```

true => x.
true => a | b.
a => c | d.

```

$a \Rightarrow \text{goal}.$
 $x, b \Rightarrow \text{goal}.$

The QEDF procedure applied to this example might yield proof pictured in Fig. 6.

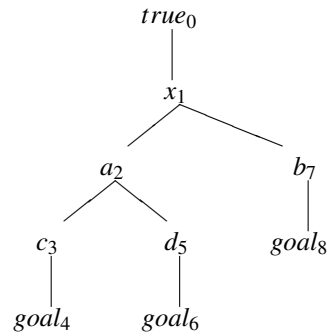


Figure 6: irrelevant case inference at node 2

The inference at node 2 in Fig. 6 is $a \Rightarrow c \mid d$, but this inference was irrelevant to the case proof for the leftmost branch (the first closing branch for the QEDF search). The second branch closing at node 6 need not have been expanded. However, the inference at node 1 was $true \Rightarrow a \mid b$, and this inference was relevant to a case proof closing at node 4. So the preferred proof tree would look like the one in Fig. 7.

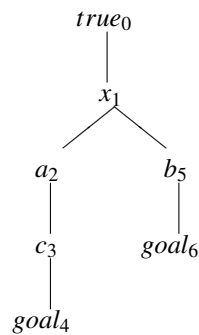


Figure 7: no irrelevant branches

When a branch saturates, the relevancy algorithm is called at the leaf and recursively computes which inferences on the branch contribute to the deduction of the leaf node: Each branch fact stores the index on the branch where that fact was inferred, and each such inference stores the antecedent of the inference, and each such antecedent has factors which store the index on the branch where they were inferred, etc. (An index of -1 signals above the root of the branch/tree).

The relevancy computation can greatly improve proof search as illustrated in Appendix A.

2 QEDF completeness issues (without function symbols)

We will discuss various versions of the QEDF algorithm, depending upon which rules of our theory are used in the queue.

Theorem 1. Suppose that the existential rules are queued. Assume that C is a valid coherent logic theory without function symbols. Then the QEDF algorithm will build a successfully saturated geolog tree for C .

We delay the proof of Theorem 1 until after a rather detailed discussion of exactly how it is that the queue is to be used in a proof search. In order to prove Theorem 1 we need to assume that the queue of existential rules is fairly used along every branch of the search tree. This means that when the search tree splits each subsequent branch search continues with the state of the queue determined at the branching point. We will refer to this as the *fair-use principle* which amounts to an assumption that each branch sees the queue for its own use, without reordering or alteration caused by the other branches.

```

true => p(a) .
s(X,Y) => goal .
r(X,Y) => p(Y) .
p(X) => u(X) | v(X) .
u(X) => s(X,A) . // A?
v(X) => r(X,B) . // B?
p(X) => s(X,Z) . // Z?
    
```

Figure 8: Sample theory (for fair-use of queue)

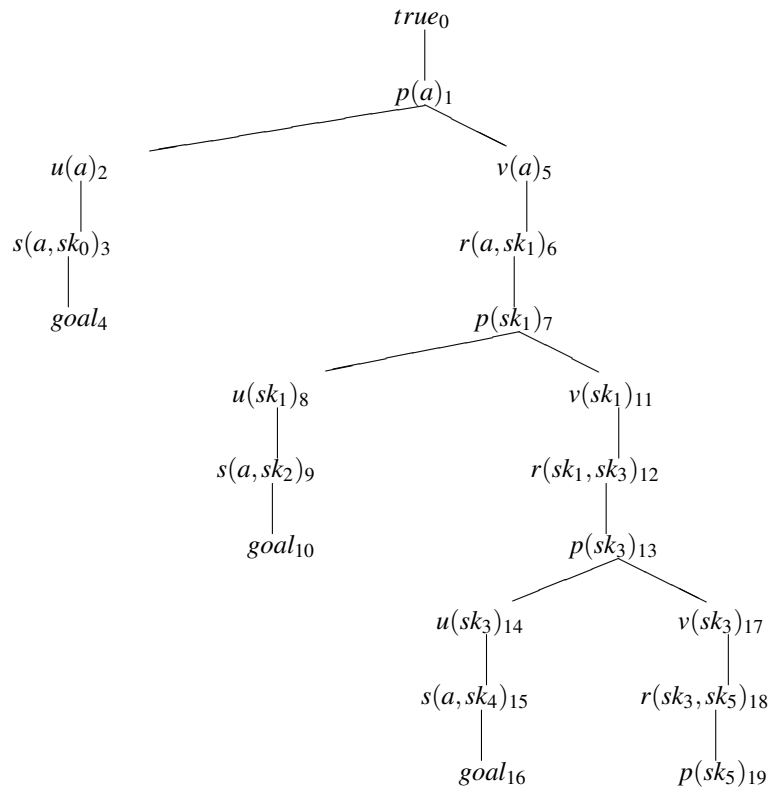


Figure 9: Infinite tree with unfair queue descends below fact #19 ...

On a visit to UiB in October 2009, Andrew Polonsky noticed the need for fair-use. Fig. 8 shows a little coherent theory that QEDF would not prove, if the queue is reset to its original order after finishing a branch and returning to the branch point to continue the search. Fig. 9 shows part of the infinite tree produced when QEDF works on the sample theory without fair-use. The tree descends infinitely downward towards the right in an apparent fashion. The same search tree is also produced if the queue is just left as it was when the previous branch saturates. The picture of the proof attempt illustrates how

interaction between the disjunctive rule and the queued existential rules effectively blocks the use of the last existential rule. Notice that the last rule only gets used at fact $u(a)_2$. (Simple depth-first search of the original theory without queueing would also *not* produce a proof.)

Now consider the same theory, but let us employ the the fair-use assumption for the queue. We get a proof as shown in Fig. 10. If we modify the rule order, as shown in Fig. 11, where the last rule is moved up in original order, then QEDF produces the proof tree depicted in Fig. 12. The simplest geolog proof tree is shown in Fig. 13, but this proof cannot be attained using QEDF even if we allow any rule reordering (why?). If the fifth rule $u(X) \Rightarrow s(X, A)$ is erased from the theory, then the behaviors described above remain exactly the same.

The example may seem somewhat artificial but the intent is to emphasize the importance of the fair-use principle governing the queue.

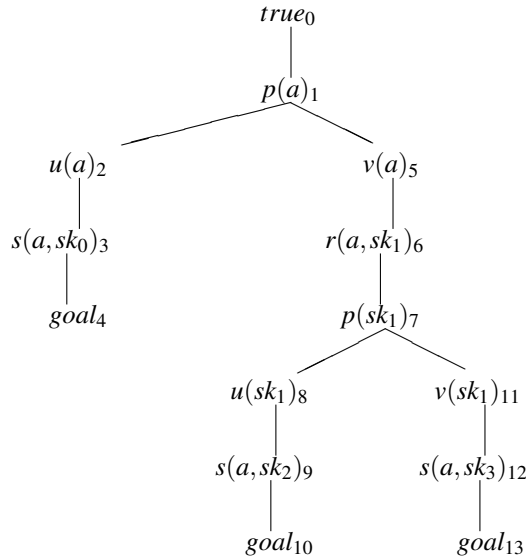


Figure 10: Proof with fair-use of queue

```

true => p(a) .
s(X,Y) => goal .
r(X,Y) => p(Y) .
p(X) => u(X) | v(X) .
p(X) => s(X,Z) . // Z?   N.B.
u(X) => s(X,A) . // A?
v(X) => r(X,B) . // B?
    
```

Figure 11: Reordered theory

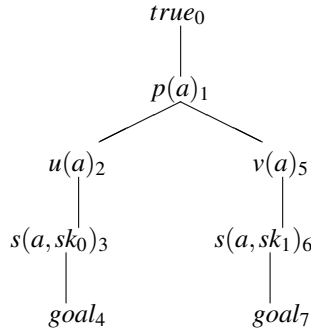


Figure 12: QEDF proof tree

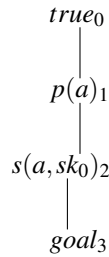


Figure 13: Simplest DF proof

Proof of Theorem 1: Suppose that T is a valid coherent logic theory, so that T has a successfully saturated complete geolog tree as explained in reference [4](Theorem 2). Suppose that QEDF does not build a successfully saturated geolog tree, using the fair-use principle for the queue. Either some QEDF search branch saturates unsuccessfully (and finitely) or else an infinite branch is generated. However, in either case, there must be at least one existential rule inference that was not used in the QEDF search that was available in the complete geolog tree, which is not possible since QEDF uses earliest-first rule satisfaction for queued rules and a fair-use regimen on every branch. \square

3 Extending QEDF for proof completeness with function symbols

Simple examples serve to show that the presence of function symbols in geolog rules could spoil completeness of QEDF, as formulated up to this point. A rule such as $\text{dom}(X) \Rightarrow \text{dom}(f(X))$, since it is a definite rule, could produce infinitely many consequences if it could apply just once.

If $p(t_1, \dots, t_k)$ is a geolog predicate, its *operator complexity* is defined to be the maximum operator complexity of its terms t_1, \dots, t_k . The operator complexity of a term is defined to be the *maximum depth of function nesting* which occurs in it. For example, the operator complexity of term $((X*Y)*Z)$ is 2, and the operator complexity of term $f(s(a), t(b))$ is 2, whereas the operator complexity of the term $f(a, b)$ is 1. Constants have term complexity 0. The operator complexity of predicate $X \succ (X*X)$ is 1.

Henceforth, let us use the term *complexity* as short for operator (function) complexity. We define the complexity of a conjunction of geolog predicates to be the maximum complexity of the conjuncts. Similarly, the complexity of a disjunction of conjunctions is the maximum complexity of any of the disjuncts. And, finally, we say that a geolog rule is *complex* provided that the complexity of its antecedent is strictly less than the complexity of its consequent. Fig. 14 shows some examples.

As a first step toward proof completeness for theories with function symbols, let us assume that QEDF includes both existential rules and complex rules. Fig.15 shows a theory with two complex rules

```

dom(X) => dom(f(X)).      // complex
p(g(X)) => p(f(X)).      // NOT complex
p(g(X)) => p(f(s(X))).   // complex
p(f(X),Y) => p(f(Y),f(X)). // NOT complex

```

Figure 14: Examples

and Fig.16 illustrates the effect that QEDF would have on a proof search for the theory in Fig. 15 if we also queue the complex rules. (This theory has no existential rules.) In the tree on the left, we see the infinite descent caused by repeated application of the second rule. Of course, it would be possible to reorder the rules in this example to achieve a proof using DF (simple depth first) without queuing, but in general reordering rules is complicated and problematic. By contrast, the modified QEDF search easily finds the proof shown in the right tree of Fig. 16.

```

true => dom(a) , dom(b) .
dom(X) , dom(Y) => dom((X*Y)) . // complex
dom(X) , dom(Y) => (X*Y) = (Y*X) . // complex
(a*b) = (b*a) => goal .

```

Figure 15: Theory with complex rules

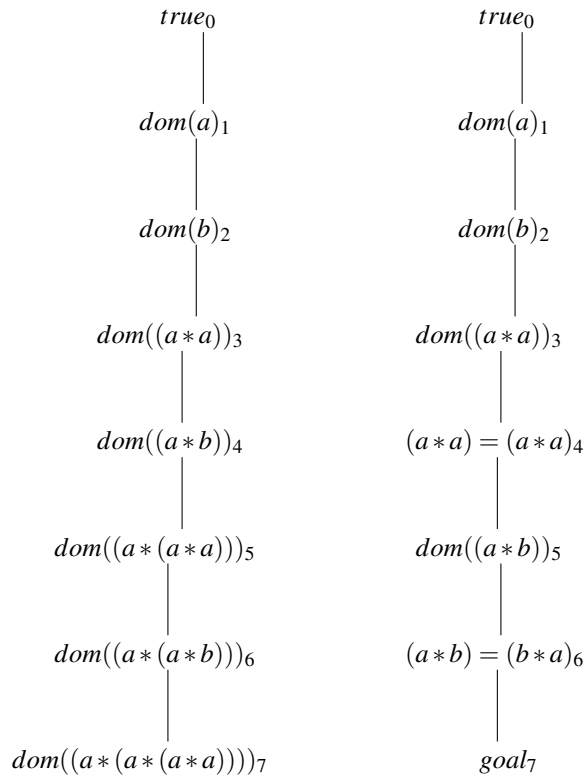


Figure 16: Compare DF (infinite descent) with QEDF (terminated) searches

However, just including complex rules in the queue will not provide proof completeness, as the

simple theory in Fig.17 demonstrates and its infinite search tree (partially) displayed in Fig.18. The problem is that the second rule is not complex, but it can – all by itself – introduce an infinite number of terms. A reordering of the theory would again solve the problem, but we are looking for a strategy where reordering (and its problems) are not necessary.

```

true => p(f(a),b) .
p(f(X),Y) => p(f(Y),f(X)) . // NOT a complex rule
p(f(X),f(Y) => goal .
    
```

Figure 17: Theory with no complex rules

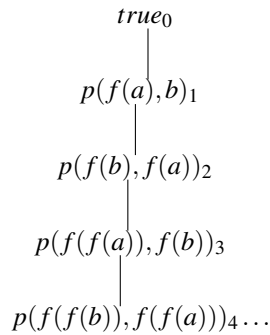


Figure 18: Not complex but still infinite ...

Notice that the rule $p(f(X),Y) \Rightarrow p(f(Y),f(X))$, although not a complex rule according to the definition, does involve a cycle ($p \Rightarrow p$) involving function symbols. We will use this as a basis for another queueing strategy.

Given a geolog theory, a *rule cycle* $\phi_1 \rightsquigarrow_{r_1} \phi_2 \rightsquigarrow_{r_2} \dots \rightsquigarrow_{r_{k-1}} \phi_k$ consists of a sequence of rules r_1, r_2, \dots, r_{k-1} of the theory such that predicate ϕ_i occurs in the antecedent of the corresponding rule r_i and ϕ_{i+1} occurs in the consequent of the rule, for $i = 1 \dots k - 1$ and $\phi_1 = \phi_k$. (The rules themselves might contain other predicates and may be disjunctive, in which case a corresponding ϕ_i would occur in at least one disjunct.) The cycle is *functional* provided that at least one of the rules has a function symbol in the consequent predicate expression, in which case the rule itself is also said to be functional.

Notice that a complex rule having a common predicate in both antecedent and consequent forms a one-rule functional rule cycle all by itself. Similarly, the rule $p(f(X),Y) \Rightarrow p(f(Y),f(X))$ forms a one-rule functional cycle.

Functional rule cycles can be somewhat complicated. Fig.19 shows a theory which has several complicated functional rule cycles.

```

true => p(a), r(b) . %1
p(f(f(X))) => goal . %2
r(X), p(Y) => s(f(X),Y) | p(X) . %3
s(f(X),Y) => s(f(Y),f(X)) . %4
s(X,Y) => p(Y) . %5
    
```

Figure 19: Theory having complicated functional rule cycles

The cycles are displayed in Fig.20. Notice that there are no existential rules in this coherent theory. The notation $p \rightsquigarrow_r q$ means that rule r has predicate p in the antecedent and predicate q in the consequent.

$$\begin{aligned}
 & s \rightsquigarrow_4 s \\
 & p \rightsquigarrow_3 s \rightsquigarrow_4 s \rightsquigarrow_5 p \\
 & p \rightsquigarrow_3 s \rightsquigarrow_5 p \\
 & p \rightsquigarrow_3 p
 \end{aligned}$$

Figure 20: Cycles

We want to consider queuing strategies where all functional rule cycles are *broken by the queue*. By this we mean that for every possible functional rule cycle, at least one of the functional rules belongs to the queue. For the example given in Fig.19, this means that $Q = [3,4]$ would break all functional cycles displayed in Fig.20. Fig.21 displays a counter model generated when either $Q = [3,4]$ or $Q = [4]$; without queuing an infinite leftmost branch is generated. However, generally speaking, this strategy of breaking functional rule cycles is not complete for generating finite counter models. An example would be the theory $\{\text{true} \Rightarrow a(1) \mid b. \quad a(X) \Rightarrow a(f(X))\}$. Please see the concluding remarks regarding the generation of countermodels.

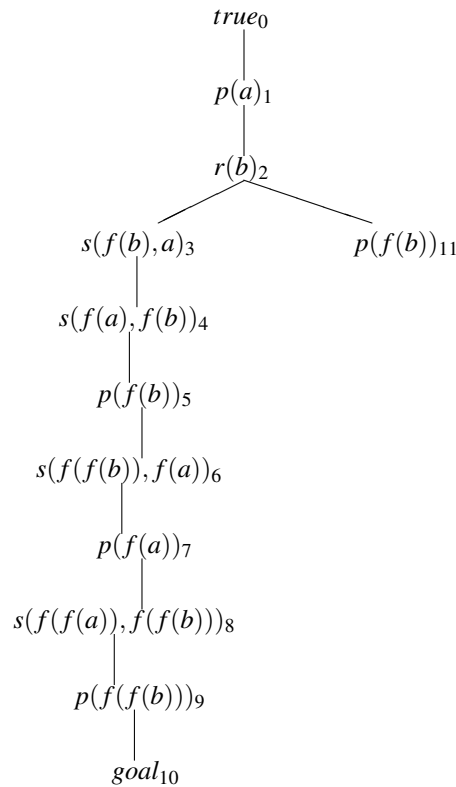


Figure 21: $Q = [3,4]$ or $Q = [4]$

There are many possible ways to break functional rule cycles. We assume that existential rules are queued, so if one encounters a functional cycle having an existential rule, that cycle is (already) broken.

Conjecture. Suppose that the existential rules are queued, and that the queue breaks all functional rule

cycles for a valid coherent logic theory C. Then the QEDF algorithm will build a successfully saturated geolog tree for theory C.

A proof of this conjecture would seem to involve essentially the same argument given for Theorem 1: The queueing strategy along with earliest-first satisfaction and application of the queued rules will eventually successfully saturate every branch, assuming that the coherent theory is valid (i.e., the theory has a successfully saturated complete geolog tree).

4 Afterwords and Conclusions

The idea regarding the queueing of existential rules was specifically proposed by Marc Bezem. John Fisher had previously suggested a similar *yielding* strategy. The need for earliest-first rule applications was suggested by John Fisher, who found counter examples when it was not used, and Marc Bezem coined the term *earliest-first* to describe the required satisfaction regimen. John Fisher and Andrew Polonsky had many discussions between March 2008 and October 2009 regarding how to extend QEDF to coherent logic including functions. Those extensions are not yet complete but this technote suggests a tentative proposal.

We have characterized the QEDF search algorithm and shown that it is complete using strategies that determine which rules to place in the queue. The acronym "QEDF" means: *Queue* certain rules, use *Earliest-first* satisfaction for queued rules and *Depth First* overall search.

It needs to be emphasized that, even if there is some finite counter-model for a coherent theory, a sequential QEDF search as described here might not find it. Consider the simple theory $\{\text{true} \Rightarrow a(1) \mid b. \quad a(X) \Rightarrow a(f(X))\}$. The sequential QEDF search will occupy its time descending under $a(1)$ and will not find the finite counter-model $\{b\}$. In order to discover finite counter-models, one needs specialized search strategies. Such strategies will be deferred to a later technical note.

The \LaTeX tree pictures in this document were automatically generated by the Colog prover [1].

References

- [1] Automated Coherent Logic Project at UiB, <http://JohnRFisher.NET/>.
- [2] 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.
- [3] 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.
- [4] John Fisher and Marc Bezem, Skolem Machines, *Fundamenta Informaticae*, 91 (1) 2009, pp.79-103.
- [5] Andrew Polonsky, *Proofs, Types, and Lambda Calculus*, Ph.D. thesis, University of Bergen, Norway, December 4, 2010.
- [6] TPTP, COM003+3.p, <http://www.cs.miami.edu/~tptp/cgi-bin/SeeTPTP?Category=Problems&Domain=COM&File=COM003+3.p>

[last update April 1, 2011]

A relevancy example

Negative-normal translations of first-order logic theories into geolog can be effectively optimized using contrapositive analysis. See Andrew Polonsky's thesis [5] regarding this topic. However, the negative-

normal translations often lead to geolog theories whose effective proofs can be made much more efficient using the relevancy algorithm discussed in Sect. 1.1.

To illustrate, consider one of Polonsky's translations of the TPTP problem COM003+3.p, which is a FOL formulation for "The Halting problem is undecidable" [6].

```

true => dom(good), dom(bad). // not needed for domain closure
tPROGRAM(V1), fPROGRAM(V1) => false.
tHALTS3(V1,V2,V3), fHALTS3(V1,V2,V3) => false.
tHALTS2(V1,V2), fHALTS2(V1,V2) => false.
tOUTPUTS(V1,V2), fOUTPUTS(V1,V2) => false.
tAnd_9(Y,Z,W), fHALTS3(W,Y,Z) => false.
tAnd_13(Y,Z,W), fHALTS3(W,Y,Z) => false.
tAnd_30(Y,Z,W), tHALTS2(Y,Z) => false.
tAnd_17(Y,Z,W) => tOr_11(Y,Z,W), tOr_15(Y,Z,W).
tOr_11(Y,Z,W), tPROGRAM(Y), tHALTS2(Y,Z) =>
    tAnd_9(Y,Z,W), tOUTPUTS(W,good).
tOr_24(Y,Z,W), tOUTPUTS(W,good) =>
    fHALTS3(W,Y,Z).
tOr_28(Y,Z,W), tOUTPUTS(W,bad) =>
    fHALTS3(W,Y,Z).
tAnd_41(W,Y,V) =>
    tOr_35(W,V,Y), tOr_39(W,Y,V).
tOr_35(W,V,Y), tPROGRAM(Y), tOUTPUTS(W,good), tHALTS2(V,Y) =>
    fHALTS3(W,Y,Y).
tOr_15(Y,Z,W), tPROGRAM(Y) =>
    tHALTS2(Y,Z); tAnd_13(Y,Z,W), tOUTPUTS(W,bad).
tOr_32(Y,Z,W) =>
    tAnd_26(Y,Z,W), tPROGRAM(Y), tHALTS2(Y,Z), tOr_24(Y,Z,W) ;
    tAnd_30(Y,Z,W), tPROGRAM(Y), tOr_28(Y,Z,W).
tOr_39(W,Y,V), tPROGRAM(Y), tOUTPUTS(W,bad) =>
    fHALTS3(W,Y,Y); tAnd_37(Y,V), tHALTS2(V,Y), tOUTPUTS(V,bad).
dom(Y), dom(Z), tForall_19(W) =>
    tAnd_17(Y,Z,W).
dom(Y), tForall_43(W,V) =>
    tAnd_41(W,Y,V).
true =>
    dom(W), tPROGRAM(W), tForall_19(W).
tPROGRAM(W) =>
    dom(Y), dom(Z), tOr_32(Y,Z,W) ;
    dom(V), tPROGRAM(V), tForall_43(W,V).

```

Figure 22: rhp.20.in

Using a QEDF prover with no relevancy check yields a proof:

```
#inferences=12310, #facts=21379, #branches=1920, time=159 ms
```

And, with the relevancy check:

```
#inferences=2170, #facts=3683, #branches=381, time=26 ms
```

This means that there were $1920 - 381 = 1539$ irrelevant case branches explored in the proof search without the relevancy check. (The data reflects two versions of the colog prover running on a PowerMac 12-core machine.)