Guarded Coherent Logic Domains

J.Fisher

Summer 2009

Abstract

This note describes a method for guarding domain closure rules that result from translation of a first-order logic theory to a coherent logic theory. The purpose for the guarding is to reduce the proof search space. The guarding information restricts the apparent domains of variables and functions. Some background references are given regarding relationships with type inference and reification. Working examples are used to explain how the guarding information is generated. An algorithm is outlined for generating rule guards. The algorithm provides an implicit typing regimen based on apparent logical domains.

1 Background

The conventional meaning of *type inference* involves the deduction of the type of a value based on its context in a programming language. Usually the programming language has declarations of profiles for functions, procures and expressions that explicitly specify what types of values are allowed in argument or parameter positions.

The type of a value in other instances of expressions, statements or forms is *implicitly* inferred based upon the positional occurrence of the value in the expression, statements or forms. See the reference [1] for several typical examples, including Hindley-Milner type inference using logical form unification.

However, in other ways the proposed method for domain guarding more closely resembles approaches for reification, which loosely involves representing implicit data of a programming language as explicit data to be used by (or in) the program. See [2] for this.

Now, when translating FOL to coherent logic, the need arises to represent universal rules using domain predicates whose scope is the entire coherent rule. In addition, the need also arises to declare domains for existential variables (or constants) which arise in the consequent of a coherent rule. See [8] for other background about translating.

This note concerns the relationship between these existential domains and the universal domains. We propose to *guard* the existential domains by specifying how they infer the universal domains. The guards amount to additional theory rules that are added based upon analysis of the coherent theory. Apparently, this approach does not require full unification of logical forms, but instead uses a simpler regimen of form *matching*. This aspect has not been completely analyzed at this writing.

The approach outlined in these notes provides a *static* analysis of the logic input theory in order to generate new guarding rules for the theory. The guarding program is bundled with a coherent logic prover, but the proof engine itself (Colog in this instance) is not modified.

Several worked examples are used to motivate our approach.

2 An example

Figure 1 displays a simple FOL theory that will be used to exemplify the methods for domain guarding. The reader should check that the conjecture is a logical consequence of the axioms and independent from the first axiom. As a preview for what lies ahead, Figure 2 provides some suggestions regarding implied domain information in the FOL theory.

```
axiom:fol:axiom_1:
![X,Y]:(c(X,Y) | d(X,Y)).

axiom:fol:axiom_2:
![X,Y]:(a(X,Y) | b(X,Y)).

axiom:fol:axiom_3:
![X,Y]:(a(X,Y) => g(X,Y)).

axiom:fol:axiom_4:
![X,Y]:(b(X,Y) => g(X,Y)).

conjecture:fol:to_prove:
![A,B]:g(A,B).
```

Figure 1: Sample FOL theory

- The negation of the conjecture would make an implied declaration that there are values A and B, where A would be the type of value that could occur as the first argument of predicate g (implied type $g/1$), and B could occur as second argument of g (implied type $g/2$) in such a way as to make $g(A,B)$ false .
- Axiom 2 declares that, for all values X that could be type $a/1$ (or $b/1$), and values Y that could be type $a/2$ (or $b/2$) one does have $(a(X,Y)|b(X,Y))$.
- Axiom 3 also implicitly types X and Y, AND also declares that implied type $g/1$ is also type $a/1$, and implied type $g/2$ is also type $a/2$. Similarly for Axiom 4.

Figure 2: Implicit typing suggestions . . .

When *Colog* translates the FOL theory into a coherent theory we obtain the coherent theory displayed in Figure 3. In this translation, the domain closure predicate is called `dom`. The translation was automatically generated by the GUI version of *Colog*, the QEDF prover described in [6].

```
/*1*/ c(X0,X1), '-c'(X0,X1) => false.
/*2*/ d(X0,X1), '-d'(X0,X1) => false.
/*3*/ a(X0,X1), '-a'(X0,X1) => false.
/*4*/ b(X0,X1), '-b'(X0,X1) => false.
/*5*/ g(X0,X1), '-g'(X0,X1) => false.
/*6*/ true => dom(A), dom(B), '+{~g(A,B)}'(A,B).
/*7*/ '+{~g(A,B)}'(A,B) => '-g'(A,B).
/*8*/ dom(X), dom(Y) => '+{(c(X,Y) | d(X,Y))}'(X,Y).
/*9*/ '+{(c(X,Y) | d(X,Y))}'(X,Y) => c(X,Y) | d(X,Y).
/*10*/ dom(X), dom(Y) => '+{(a(X,Y) | b(X,Y))}'(X,Y).
/*11*/ '+{(a(X,Y) | b(X,Y))}'(X,Y) => a(X,Y) | b(X,Y).
/*12*/ dom(X), dom(Y) => '+{(a(X,Y) => g(X,Y))}'(X,Y).
/*13*/ '+{(a(X,Y) => g(X,Y))}'(X,Y) => '-a'(X,Y) | g(X,Y).
/*14*/ dom(X), dom(Y) => '+{(b(X,Y) => g(X,Y))}'(X,Y).
/*15*/ '+{(b(X,Y) => g(X,Y))}'(X,Y) => '-b'(X,Y) | g(X,Y).
```

Figure 3: Translated coherent theory, large.gl

Figure 4 shows the result of a proof search for the coherent theory of Figure 3. This little FOL theory has a surprisingly large proof! The graphical prover reveals that the search tree contains all possible combinations of domain values, most of which are completely irrelevant, and even attempts to mix in facts generated by domain closures from the first axiom. It is quite a fast proof, but still a very large search space.

```
$ java -jar Colog.jar ./gl/large.gl 100
...
Colog1.0, file=./gl/large.gl, PROOF, #inferences=6725, #facts=8678, time=194ms
```

Figure 4: A LARGE proof.

We note that there are other translations into geolog theory that would have smaller proofs, but this translation is chosen here to illustrate the point of domain guarding.

3 Adding domain guards

The Colog prover has a utility for adding domain guards, with the usage illustrated in Fig. 5. The domain name in the theory is passed as the last command-line argument.

```
$ java -cp Colog.jar Guard ./gl/large.gl  dom
{.. writes new file ./gl/large.gl.gd .. }
```

Figure 5: Usage for Guard utility.

Figure 6 shows the coherent theory (`large.gl.gd`)) after adding domain guards.

```
/*1*/ c(X0,X1), '-c'(X0,X1) => false.
/*2*/ d(X0,X1), '-d'(X0,X1) => false.
/*3*/ a(X0,X1), '-a'(X0,X1) => false.
/*4*/ b(X0,X1), '-b'(X0,X1) => false.
/*5*/ g(X0,X1), '-g'(X0,X1) => false.
/*6*/ true => dom1(A), dom2(B), '+{~g(A,B)}'(A,B).
/*7*/ '+{~g(A,B)}'(A,B) => '-g'(A,B).
/*8*/ dom3(X), dom4(Y) => '+{(c(X,Y) | d(X,Y))}'(X,Y).
/*9*/ '+{(c(X,Y) | d(X,Y))}'(X,Y) => c(X,Y) | d(X,Y).
/*10*/ dom5(X), dom6(Y) => '+{(a(X,Y) | b(X,Y))}'(X,Y).
/*11*/ '+{(a(X,Y) | b(X,Y))}'(X,Y) => a(X,Y) | b(X,Y).
/*12*/ dom7(X), dom8(Y) => '+{(a(X,Y) => g(X,Y))}'(X,Y).
/*13*/ '+{(a(X,Y) => g(X,Y))}'(X,Y) => '-a'(X,Y) | g(X,Y).
/*14*/ dom9(X), dom10(Y) => '+{(b(X,Y) => g(X,Y))}'(X,Y).
/*15*/ '+{(b(X,Y) => g(X,Y))}'(X,Y) => '-b'(X,Y) | g(X,Y).

dom1(Z) =>  dom5(Z), dom7(Z), dom9(Z).
dom2(Z) =>  dom6(Z), dom8(Z), dom10(Z).
```

Figure 6: large.gl.gdd

Notice that each `dom` has been uniquely indexed. Dom's that appear in the consequent of a rule are called *existential* doms and those appearing in the antecedent are called *universal* doms.

Each dom determines an *orbit* of implicit types, which can be statically determined from the rules of the theory. For example, Figure 7 displays the orbits for `dom1` and `dom5`.

```
dom1: { '+{~g(A,B)}'/1, '-g'/1, g/1, '-a'/1, a/1, '-b'/1, b/1 }

dom5: { '+{(a(X,Y) | b(X,Y))}'/1, a/1, b/1, '-a'/1, '-b'/1, g/1, '-g'/1 }
```

Figure 7: Orbits of dom1 and dom5

Since these orbits intersect, we add the forwarding rule `dom1(Z) => dom5(Z)`. The reason for this is that the orbit calculation verifies that a value which arises from the existential dom could, if provided the occasion, be a value to which the universal dom should refer. Other orbits are calculated similarly. The reader can check that dom1 and dom3 have mutually exclusive orbits, so there is no domain forwarding in that case.

If the orbit of existential *domi* has nonempty intersection with the orbit of universal *domj*, then add the forwarding rule `domi(Z) => domj(Z)`.

Figure 8 displays the result of a proof search for the guarded domain theory. The effect of guarding reduced the size of the search tree by 99.7% – an excellent improvement. Figure 9 displays the graphical proof tree (automatically generated LaTeX code using the Colog GUI).

```
$ java -jar Colog.jar ./gl/large.gl.gdd 100
...
Colog1.0, file=./gl/little.gl, PROOF, #inferences=17, #facts=27, time=1ms
```

Figure 8: Proof using guarded theory

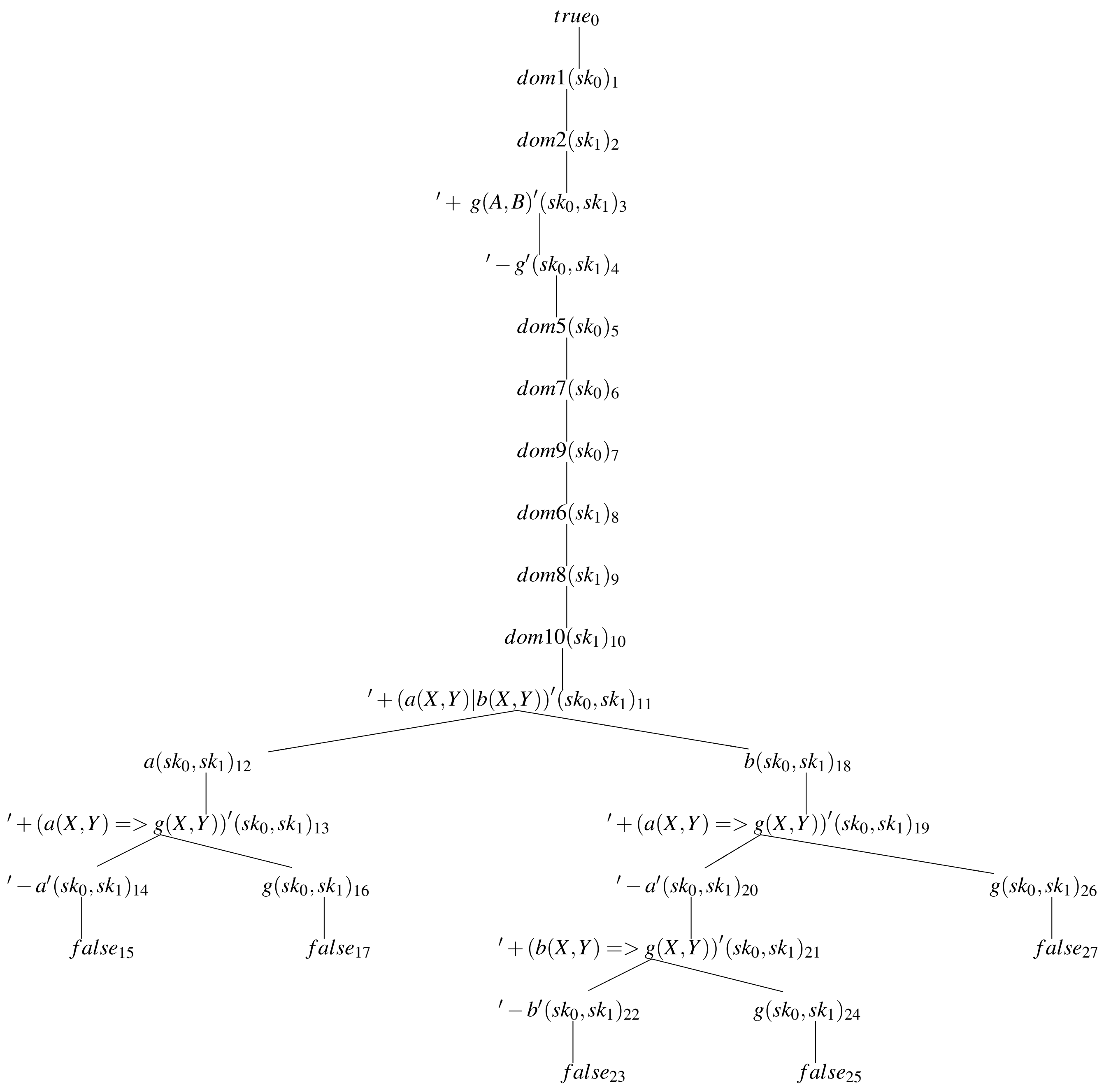

Figure 9: The small proof tree

4 Guarding function closures

We use another example, shown in Fig. 10, to explain how guarding can work when there are function closure rules. Notice that functions `a` and `f` only appear in place `p/1`, and that functions `b` and `g` only occur in place `p/2`.

When Colog translates `fc.fol` into a geolog theory (preserving coherent rules this time), we get the geolog theory shown in Fig. 11. Colog can prove this theory using 78 inferences, many of them irrelevantly using functions in unguarded doms.

Fig. 12 shows the guarded domain theory. Notice that the functions closures (rules #9, #10) only close for places where the functions actually appeared in the original theory.

```
axiom:fol:ax_1:
p(f(a),g(b)).

axiom:coherent:ax_2:
![X,Y]:( p(X,Y) => q(Y,X) ) .

conjecture:fol:conjecture:
?[X,Y]: q(X,Y).
```

Figure 10: Theory `fc.fol` with functions

```
/*1*/ true => dom(a).
/*2*/ true => dom(b).
/*3*/ p(X0,X1), '-p'(X0,X1) => false.
/*4*/ q(X0,X1), '-q'(X0,X1) => false.
/*5*/ dom(X), dom(Y) => '+{~q(X,Y)}'(X,Y).
/*6*/ '+{~q(X,Y)}'(X,Y) => '-q'(X,Y).
/*7*/ true => p(f(a),g(b)).
/*8*/ p(X,Y) => q(Y,X).
/*9*/ dom(X0) => dom(f(X0)).
/*10*/ dom(X0) => dom(g(X0)).
```

Figure 11: Unguarded geolog theory `fc.gl`

```
$ java -cp Colog.jar Guard ./gl/large.gl  dom
                  ...

                /*1*/ true => dom1(a).
                /*2*/ true => dom2(b).
                /*3*/ p(X0,X1), '-p'(X0,X1) => false.
                /*4*/ q(X0,X1), '-q'(X0,X1) => false.
                /*5*/ dom3(X), dom4(Y) => '+{~q(X,Y)}'(X,Y).
                /*6*/ '+{~q(X,Y)}'(X,Y) => '-q'(X,Y).
                /*7*/ true => p(f(a),g(b)).
                /*8*/ p(X,Y) => q(Y,X).
                /*9*/ dom1(Z) => dom4(Z).
                /*10*/ dom2(Z) => dom3(Z).
                /*11*/ dom4(X0) => dom4(f(X0)).
                /*12*/ dom3(X0) => dom3(g(X0)).
```

Figure 12: Guarded geolog theory `fc.gl.gd`

Colog provides a proof for the guarded theory having only 17 inferences, depicted in Fig. 13.

$$\begin{tikzcd}
true_0 \arrow[d, dash] \\
dom1(a)_1 \arrow[d, dash] \\
dom2(b)_2 \arrow[d, dash] \\
dom4(a)_3 \arrow[d, dash] \\
dom3(b)_4 \arrow[d, dash] \\
'+\ q(X,Y)'(b,a)_5 \arrow[d, dash] \\
'-q'(b,a)_6 \arrow[d, dash] \\
p(f(a),g(b))_7 \arrow[d, dash] \\
q(g(b),f(a))_8 \arrow[d, dash] \\
dom4(f(a))_9 \arrow[d, dash] \\
'+\ q(X,Y)'(b,f(a))_{10} \arrow[d, dash] \\
'-q'(b,f(a))_{11} \arrow[d, dash] \\
dom3(g(b))_{12} \arrow[d, dash] \\
'+\ q(X,Y)'(g(b),a)_{13} \arrow[d, dash] \\
'+\ q(X,Y)'(g(b),f(a))_{14} \arrow[d, dash] \\
'-q'(g(b),a)_{15} \arrow[d, dash] \\
'-q'(g(b),f(a))_{16} \arrow[d, dash] \\
false_{17}
\end{tikzcd}$$

Figure 13: fc.gl.gd proof

Fig. 14 depicts the saturated connection graph that Guard computes for this example. The table is square with indices (# place) shown along the left-hand edge (and which match the unlabelled column indices). The graph is initially populated with connections between places based upon variables in rules, or initial connections based upon place occurrences of functions. The saturation of this graph then effectively computes the transitive closure of the orbit relation: if place p_1 is in the orbit of place p_2 and p_2 is in the orbit of place p_3, then p_1 is in the orbit of p_3.

```
0 dom1/1            1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
1 dom2/1            0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
2 dom5/1            1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
3 dom6/1            0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
4 p/2               0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
5 '-p'/2            0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
6 p/1               1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
7 '-p'/1            1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
8 q/2               1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
9 '-q'/2            1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
10 q/1              0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
11 '-q'/1           0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
12 dom4/1           1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
13 '+{~q(X,Y)}'/2  1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0
14 dom3/1           0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
15 '+{~q(X,Y)}'/1  0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1
```

Figure 14: Saturated connection graph for `fc.gl`

The nonempty intersection of rows means that the places or predicates labeled on the left have intersecting orbits. `dom5` and `dom6` are temporary template places, corresponding respectively to function closure rules `dom5(X0) => dom5(f(X0))` and `dom6(X0) => dom6(g(X0))`. So, for example, `dom5` is a temporary predicate or place whose orbit detects occurrences of function `f`. The table shows that `dom5`'s orbit intersects that of the universal `dom4`, but not universal `dom3`, which accounts for why a closure of `f` for `dom4` is added, but not for `dom3`.

5 Optimal guards

The graph algorithm for computing intersecting orbits (described briefly at the end of the previous section) is sufficient to restrict domains to what is sufficient for a proof. That is, the algorithm is complete: the resulting guarded theory will prove provide the source unguarded theory proves.

However, some of the forwarding or function closure rules may not be necessary for proof. Here is a rather dramatic example for what can happen. Fig. 15 shows one of Andrew Polonsky's automatic translations of the TPTP problem COM003+3.p (The halting problem is undecidable). This example does not have functions.

```
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 15: rhp.20.gl

Fig. 16 shows the automatically guarded theory.

```
/*1*/ true => dom1(good), dom2(bad).
/*2*/ tPROGRAM(V1), fPROGRAM(V1) => false.
/*3*/ tHALTS3(V1,V2,V3), fHALTS3(V1,V2,V3) => false.
/*4*/ tHALTS2(V1,V2), fHALTS2(V1,V2) => false.
/*5*/ tOUTPUTS(V1,V2), fOUTPUTS(V1,V2) => false.
/*6*/ tAnd_9(Y,Z,W), fHALTS3(W,Y,Z) => false.
/*7*/ tAnd_13(Y,Z,W), fHALTS3(W,Y,Z) => false.
/*8*/ tAnd_30(Y,Z,W), tHALTS2(Y,Z) => false.
/*9*/ tAnd_17(Y,Z,W) => tOr_11(Y,Z,W), tOr_15(Y,Z,W).
/*10*/ tOr_11(Y,Z,W), tPROGRAM(Y), tHALTS2(Y,Z) => tAnd_9(Y,Z,W), tOUTPUTS(W,good).
/*11*/ tOr_24(Y,Z,W), tOUTPUTS(W,good) => fHALTS3(W,Y,Z).
/*12*/ tOr_28(Y,Z,W), tOUTPUTS(W,bad) => fHALTS3(W,Y,Z).
/*13*/ tAnd_41(W,Y,V) => tOr_35(W,V,Y), tOr_39(W,Y,V).
/*14*/ tOr_35(W,V,Y), tPROGRAM(Y), tOUTPUTS(W,good), tHALTS2(V,Y) => fHALTS3(W,Y,Y).
/*15*/ tOr_15(Y,Z,W), tPROGRAM(Y) => tHALTS2(Y,Z) | tAnd_13(Y,Z,W), tOUTPUTS(W,bad).
/*16*/ 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).
/*17*/ tOr_39(W,Y,V), tPROGRAM(Y), tOUTPUTS(W,bad) => fHALTS3(W,Y,Y) |
                                   tAnd_37(Y,V), tHALTS2(V,Y), tOUTPUTS(V,bad).
/*18*/ dom3(Y), dom4(Z), tForall_19(W) => tAnd_17(Y,Z,W).
/*19*/ dom5(Y), tForall_43(W,V) => tAnd_41(W,Y,V).
/*20*/ true => dom6(W), tPROGRAM(W), tForall_19(W).
/*21*/ tPROGRAM(W) => dom7(Y), dom8(Z), tOr_32(Y,Z,W) |
                                  dom9(V), tPROGRAM(V), tForall_43(W,V).
/*22*/ dom6(Z) => dom3(Z), dom4(Z), dom5(Z).
/*23*/ dom7(Z) => dom3(Z), dom4(Z), dom5(Z).
/*24*/ dom8(Z) => dom3(Z), dom4(Z), dom5(Z).
/*25*/ dom9(Z) => dom3(Z), dom4(Z), dom5(Z).
```

Figure 16: `rhp.20.gl.gd`

And, finally, Fig. 17 shows just a few modifications to the domain forwarding rules, done by hand.

```
...

/*22*/ dom6(A) => dom3(A).
/*23*/ dom7(A) => dom3(A), dom5(A).
/*24*/ dom8(A) => dom4(A), dom5(A).
/*25*/ dom9(Z) => dom3(Z), dom4(Z), dom5(Z).
```

Figure 17: By-hand modifications `rhp.20.min`

Fig. 18 shows the results of the Colog proof runs for all three versions.

```
$ java -jar Colog.jar /geolog/technotes/GuardedDomains/rhp.20.gl 200
Colog1.0, file=/geolog/technotes/GuardedDomains/rhp.20.gl, PROOF,
         #inferences=12310, #facts=21379, time=1268ms

$ java -jar Colog.jar /geolog/technotes/GuardedDomains/rhp.20.gl.gd 200
Colog1.0, file=/geolog/technotes/GuardedDomains/rhp.20.gl.gd, PROOF,
         #inferences=625, #facts=1158, time=60ms

$ java -jar Colog.jar /geolog/technotes/GuardedDomains/rhp.20.min 200
Colog1.0, file=/geolog/technotes/GuardedDomains/rhp.20.min, PROOF,
         #inferences=74, #facts=136, time=13ms
```

Figure 18: Proofs

We believe that the shortest proof in Fig. 18 is a minimal proof, based upon inspection of the visualizer/GUI version of Colog proof. (In summary, that visualization displays a proof tree with no repeated subtrees!)

At present, we do not see any static pattern in the theory itself that would predict why some of the domain forwarding could be reduced. But the example illustrates that the connection graph approach may produce many more than necessary guards for proof, even though it may often reduce proofs dramatically.

The algorithm for computing function closures seems somewhat naive, and can possibly be refined further.

References

[1] Wikipedia article *Type_inference*, `http://en.wikipedia.org/wiki/Type_inference`

[2] Wikipedia article *Reification (computer science)*, `http://en.wikipedia.org/wiki/Reification_(computer_science)`

[3] 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.

[4] 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*, Universite d'Orleans - LIFO, Orleans, France September 10-14, 2007. Springer LNCS vol. 4664, pp. 182-192.

[5] John Fisher and Marc Bezem, Skolem Machines, *Fundamenta Informaticae*, 91 (1) 2009, pp.79-103.

[6] John Fisher, *QEDF Proof Search for Coherent Logic*, technical note, winter 2009.

[7] John Fisher, *Concurent Coherent Logic*, technical note, spring 2009.

[8] Andrew Polonsky and Marc Bezem, Proof Objects for Logical Translations, Proc. The 1st Coq Workshop, Munich, Germany 21 August, 2009 (49-61) `http://coq.inria.fr/files/coq-workshop-TUM-I0919.pdf`.

[9 Oct 2009]