AutoLog Equality Inference

John Fisher
Prof. Emer. Computer Science
Cal Poly Pomona
jrfisher@cpp.edu
fisher.r.john@gmail.com

BLAST 2018
August 6-10
University of Denver

Topics

Skolem Machines

Reasoning with equality

Modulated coherent logic using AutoLog

AutoLog requirements, design and implementation issues

1– Skolem Machines/Theory

- The following paper (FI) introduced the idea of a Skolem Machine for computing coherent logic.

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

  The paper gives conceptual definitions of a Skolem Machine – as a *tape* machine analagous to a Turing Machine and as a *tree* machine – and shows that it has theoretical computational universality. The paper itself is linked at the following website; the paper's references provide some background regarding the moniker "Skolem Machine". "Skolem Abstract Machine (SAM)" refers to any of the several computer implementations.

- `http://skolemmachines.org`

2– Skolem Machines/Programming Code

A *colog rule* (Skolem machine code instruction) has the general form . . .

$$A_1,\ A_2,\ \ \ldots\ ,\ A_m => C_1 \mid C_2 \mid \ \ldots\ \mid C_n.$$

The *antecedent* of the rule is a conjunction of literals A_i, the *consequent* of the rule is a disjunction of C_i, each of which is itself a conjunction of literals ($m, n \geq 1$).

3– Skolem Machines/Code Example with Proof Tree Object

The colog program (colog14)

```
true => domain(X), p(X).
p(X) => q(X) | r(X) | domain(Y), s(X,Y).
domain(X) => u(X).
u(X), q(X) => false.
r(X) => goal.
s(X,Y) => goal.
```

An associated FOCL, first-order coherent logic, theory

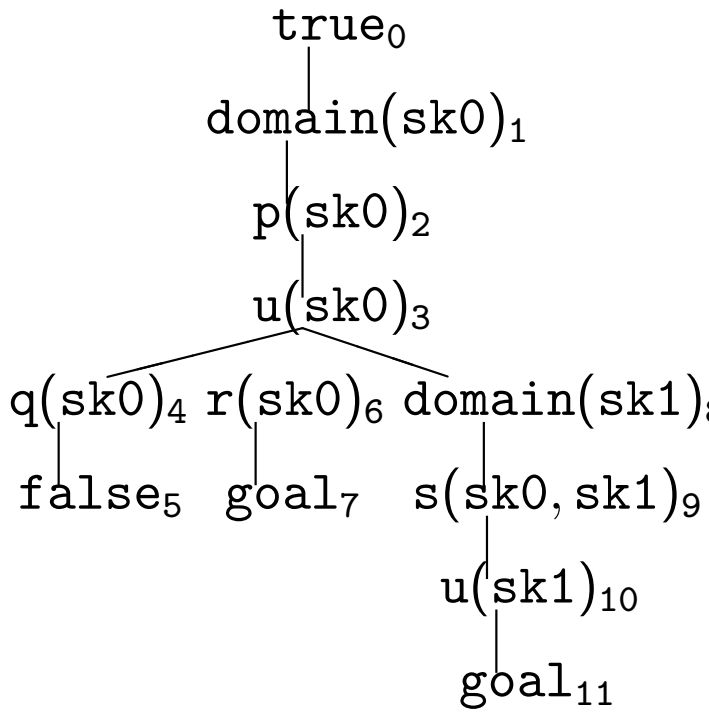
$\top \rightarrow \exists x\ p(x).$

$\forall x(p(x) \rightarrow q(x) \vee r(x) \vee \exists y\ s(x,y)).$

$\forall x\ (u(x) \wedge q(x) \rightarrow \bot).$

$- - - - - - - - - - - - - - - - - -$

$\exists x\ r(x) \vee \exists x,y\ s(x,y)\ ?$

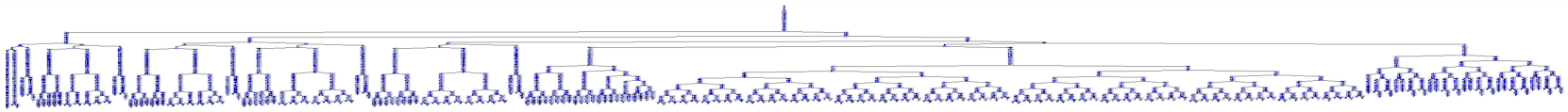


4– . . . extracted proof

```
file:/SkolemMachines.ORG/reports/BLAST18/automod/1.txt
LEAF 5.
@4, rule4: u(sk0), q(sk0) => false
@3, rule2: p(sk0) => q(sk0) | r(sk0) | domain(sk1), s(sk0,sk1)
@2, rule3: domain(sk0) => u(sk0)
@0, rule1: true => domain(sk0), p(sk0)
LEAF 7.
@6, rule5: r(sk0) => goal
@3, rule2: p(sk0) => q(sk0) | r(sk0) | domain(sk1), s(sk0,sk1)
@2, rule3: domain(sk0) => u(sk0)
@0, rule1: true => domain(sk0), p(sk0)
LEAF 11.
@10, rule6: s(sk0,sk1) => goal
@3, rule2: p(sk0) => q(sk0) | r(sk0) | domain(sk1), s(sk0,sk1)
@2, rule3: domain(sk0) => u(sk0)
@0, rule1: true => domain(sk0), p(sk0)
```

5– Skolem Machines – difficult problems, e.g.

A coherent logic formulation of the TPTP problem COM003+3.p (*The halting problem is undecidable*) was given by A. Polonsky using his translator from FOL form to various colog forms*. Various translations could all be quickly solved by colog14 (1-3 ms). The following screenshot shows a very branchy (but speedy) proof tree object (281 proof cases).



Domain guarded solutions (my experiments) were less branchy, for example (10 cases) . . .

*These translations did NOT employ intuitionistic transforms. (More on that later.)

6– Skolem Machine – constructive logic

Coherent implications form a Glivenko class: If a coherent implication is derivable from a coherent theory using classical logic, then it is derivable intuitionistically.

Any consequence provable using Skolem Machine computations for a colog program is a provable consequence of the program using intuitionistic logic (where *consequence* here is defined in the FI theory article – a DNF derived using a tree computation).

(Statements on this slide may require qualification for AutoLog. *)*

7– Reasoning with equality – example (colog14)

Modulation inflation problem for theorem provers . . .

Problem: Show that left and right inverses in a monoid are equal.

```
% data
  true => dom(e),
          dom(x), dom(y), dom(z), % hypothesis
          (y*x)=e,                % left inverse for x
          (x*z)=e.                % right inverse for x
% conjecture
  y=z => goal.
% closure for *                          C
  dom(X), dom(Y) => dom((X*Y)).
% associativity of *                     C
  dom(X), dom(Y), dom(Z) => ((X*Y)*Z)=(X*(Y*Z)).
% e is identity for *                    C
  dom(X) => (X*e)=X, (e*X)=X.
% = rules
  dom(X) => X=X.
  X=Y => Y=X.
  X=Y, Y=Z => X=Z.
% substitutivity for =                   C
  A=B, C=D => (A*C)=(B*D).
%                                 C: complex rule
```

8– monoid example (cont.)

Using a *cut* strategy (≤ 2 applications of complex rule), one gets:

`Proof cut=2 inferences=1384 facts=1407 time=22ms subsumed=18867`

Without cut, one gets:

`Proof inferences=5538 facts=5599 time=244 ms subsumed=202135`

```
@1406, rule1: y=z => goal
@1405, rule7: z=y => y=z
@1404, rule8: z=((y*x)*z), ((y*x)*z)=y => z=y
@1341, rule7: y=((y*x)*z) => ((y*x)*z)=y
@1340, rule8: y=(y*(x*z)), (y*(x*z))=((y*x)*z) => y=((y*x)*z)
@1319, rule7: ((y*x)*z)=(y*(x*z)) => (y*(x*z))=((y*x)*z)
@1318, rule4: dom(y), dom(x), dom(z) => ((y*x)*z)=(y*(x*z))
@650, rule8: y=(y*e), (y*e)=(y*(x*z)) => y=(y*(x*z))
@649, rule7: (y*(x*z))=(y*e) => (y*e)=(y*(x*z))
@648, rule9: y=y, (x*z)=e => (y*(x*z))=(y*e)
@93, rule8: z=(e*z), (e*z)=((y*x)*z) => z=((y*x)*z)
@92, rule7: ((y*x)*z)=(e*z) => (e*z)=((y*x)*z)
@91, rule9: (y*x)=e, z=z => ((y*x)*z)=(e*z)
@87, rule7: (e*z)=z => z=(e*z)
@84, rule5: dom(z) => (z*e)=z, (e*z)=z
@66, rule7: (y*e)=y => y=(y*e)
@64, rule5: dom(y) => (y*e)=y, (e*y)=y
@9, rule6: dom(z) => z=z
@8, rule6: dom(y) => y=y
@0, rule2: true => dom(e), dom(x), dom(y), dom(z), (y*x)=e, (x*z)=e
```

9– Complexity Cut mechanism

The current *complexity cut* mechanism, which is rule based, can limit (cut) the number of applications of a rule whose consequent has larger complexity than that of its antecedent. Complexity is defined in terms of the number of operators) is a term.

Rules which are not complex are NOT restricted by a cut wizard; these rules do not introduce more complicated facts onto the search branch. Only rules which introduce more complicated search facts are restricted.

Substitution of equals (modulation) involves the same complexity issue.

10– Complexity Cut mechanism (cont.)

The *complexity cut mechanism* is akin to Prolog's cut mechanism (which also cuts off rule alternatives).

The Skolem Machine complexity cut mechanism is sound.

Also, any problem which proves can be proved using some complexity cut. (Pick a cut that would allow all of the inferences required for the proof in hand.)

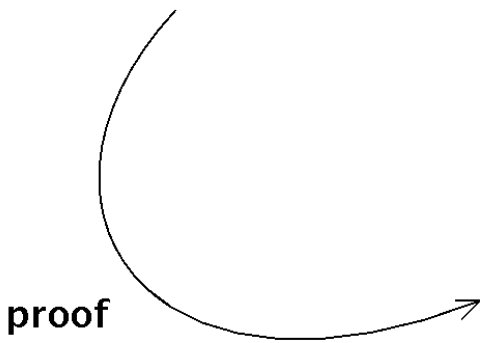
11– Modulated coherent logic using AutoLog

A new logic programming language (and theorem prover) design is currently being developed, called Autolog. Autolog employs a Skolem Machine for extended coherent logic computations. We plan to augment Autolog with appropriate renditions of equality modulation in order to enable Prover9-like capabilities for Autolog.

Example A involves the use of substructural logic equalities involving constructive negation as *lemmas* to modulate proof terms – *metalogic.* Lemmas for specific logics amount to a kind of *logical anti-cut mechanism*: Algebraic lemmas provable for specific logics are used as modulators for various coherent theories. (A *logical cut* of the lemmas would amount to a combined theory proving the lemmas and the main theorems. *Auto-provers prefer cuts (divide-and-conquer tactics), logicians require cut eliminations.*)

12– AutoLog example A: modulated proof terms

```
// equality lemma
     { ¬(A∨B)=¬A∧¬B. } // A Lemma, Heyting transform ‡
// coherent logic rules
     true => ¬(p(a) ∨ q(a)). // #1
     P∧Q => P, Q.            // #2
     ¬q(Z) => goal.          // #3
```




```
      true
        |          via #1
  ¬(p(a)∨q(a))
        |-         via lemma modulation
  ¬p(a)∧¬q(a)
        |          via #2
     ¬p(a)
        |           ...
     ¬q(a)
        |          via #3  (Z=a)
      goal
```

Proof terms can be algebraic expressions modulated (substituted) by equality lemmas (which could be weighted towards a preferred subsitution). The "bigger plan" here is to devise mechanisms for "universal logic computations", implemented using a Skolem Machine. This example illustrates how we might incorporate constructive negation into our logic computations.

13– AutoLog, monoid problem redux

It is possible to reformulate the monoid problem on slide 7 so that the identities are used at modulators:

```
(X*Y)*Z=X*(Y*Z).
X*e=X.
e*X=X.
```

The reflexivity, symmetry, transitivity and substitutivity can be internalized or left as coherent logic rules. If C is a bound equational proof literal and E is an equation (intended as a modulator), then any "paramodulant" of C by E can be achieved by a "demodulation" (i.e, a substitution). This observation simplifies the modulation requirements somewhat; however, some other details are still open.

The complexity cut mechanism should still work in a similar manner for the new formulations of equality modulation. Other efficiencies are mentioned in the last section.

14– AutoLog example B: more metalogic

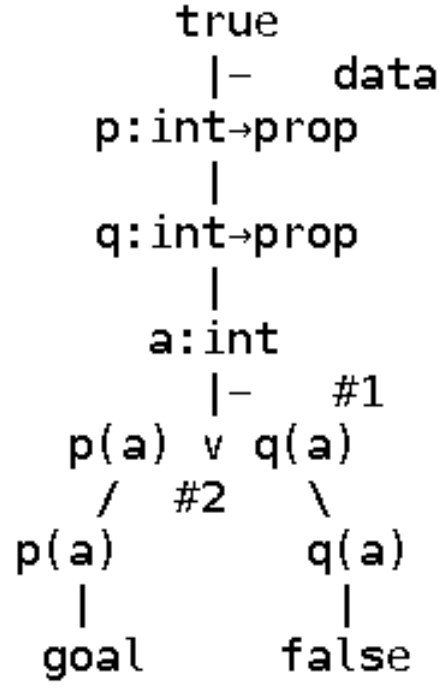
```
// rule has types, impredicativity, indexicality
   P:T→prop, Q:T→prop, X:T, P(X) => P(X)vQ(X).  // #1
// coherent reification of v
   P v Q => P | Q.                               // #2

true => p:int→prop, q:int→prop, a:int.        // data
p(a) => qoal.
q(a) => false.
```

A => A ∨ B.

will not work
as intended for #1

proof

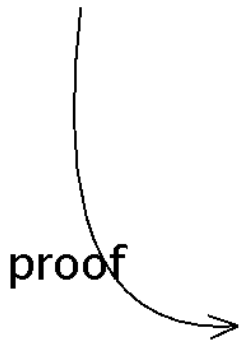


15– AutoLog example C: ∀, ∃ metalogic

```
P:T→prop, ∀(X:T,P(X)), Z:T => P(Z).   // #1

P:T→prop, ∃(X:T,P(X)) => X:T, P(X).   // #2

// -----
true => p:int→prop, ∃(X:int,p(X)).    // data
p(Z) => goal.
```




```
          true
           |-            data
     p:int→prop
           |
∃(n:int,p(n))
           |-            rule #2
       n:int
         |
       p(n)
        |-              Z=n
       goal
```

16– AutoLog example D: residuated modulators

An equality modulator $A = B$ is potentially applicable in a symmetric fashion, even though a search stategy might prefer a one-way substitution (See, e.g. slide 12). A *residuated modulator* $A \to B$ requires a proof term (branch fact) match of A replaced by the corresponding instance of B (and not vice-versa). Here are some one-way constructive negation modulators, for example:

```
P ≤ ¬¬P.
P ∧ ¬P ≤ ⊥.          ¬P ∧ ¬¬P ≤ ⊥.
A ∧ ¬B ≤ ¬(A→B).     ¬(A→B) ≤ ¬¬A ∧ ¬B.
```

Note that a form $A \leq A \vee B$ would not make sense without context–see slide 14.
The use of rules is preferred to one-side modulation.

Are there complete sets of rules and modulators for constructive negation (Heyting algebra)?

17- Unified Logic Computation: Model=Proof

The Skolem machine builds models or proofs using the same logic computations.

Model=Proof

$\bigwedge$

tree models proof trees

$\bigvee$

logic computation theory $\approx$ *procedural semantics*

via

Equality Modulated Algebraic Logic

18- Similarity with Kripke semantics – *a*

These meta-axioms can be considered as specifying how to make residuated tree models . . .

$$
\begin{aligned}
&A \wedge B \Rightarrow A,\ B.\\
&A \vee B \Rightarrow A \mid B.\\
&A \to B,\ A \Rightarrow B.\\
&\bot \Rightarrow \mathtt{false}.\\
&\{\ A \to \bot = \neg A.\ \}
\end{aligned}
$$

Consider the modulator as a *definition*.
PROJECT: Describe a cogent relationship between those tree models and Kripke semantics. How would tree branches correspond to preordered Kripke frames? (next slide)
(Note that tree branch sets contain grounded *propositions*!)

19- Similarity with Kripke semantics – *b*

An **intuitionistic Kripke model** is a triple $\langle W, \leq, \Vdash \rangle$, where $\langle W, \leq \rangle$ is a preordered Kripke frame, and $\Vdash$ satisfies the following conditions:

- if p is a propositional variable, $w \leq u$, and $w \Vdash p$, then $u \Vdash p$ (*persistency* condition (cf. monotonicity)),
- $w \Vdash A \wedge B$ if and only if $w \Vdash A$ and $w \Vdash B$,
- $w \Vdash A \vee B$ if and only if $w \Vdash A$ or $w \Vdash B$,
- $w \Vdash A \to B$ if and only if for all $u \geq w$, $u \Vdash A$ implies $u \Vdash B$,
- not $w \Vdash \bot$.

The negation of A, $\neg A$, could be defined as an abbreviation for $A \to \bot$. If for all u such that $w \leq u$, not $u \Vdash A$, then $w \Vdash A \to \bot$ is vacuously true, so $w \Vdash \neg A$.

Intuitionistic logic is sound and complete with respect to its Kripke semantics, and it has the finite model property.

clipped from ...

`https://en.wikipedia.org/wiki/Kripke_semantics#Semantics_of_intuitionistic_logic`

20– AutoLog USM challenge

Is there an AutoLog formulation for a *Universal Skolem Machine*? Using algebraic and metalogic formulations – machine simulating self?

We know that the SM is a universal computer, using the adequacy condition in Marvin Minsky's famous paper. (See §5 of FI.)

21- AutoLog requirements, design and implementation issues

Most of the new design requirements lead to implementation requirements that force new efficiencies on the machine computations. That aspect of AutoLog is a kind of *compiler optimization task*: Logic code must be translated and optimized for logic machine internal code computation.

- 1 impredicativity vs indexicality (e.g. slide 14)
- 2 tabled logic, term indexing
- 3 auto-confluent modulation reasoning (?)
- 4 ADT (algebraic data type) reasoning
- 5 staged rule application
- 6 concurrent staged rule applications
- 7 modules or lemmas
- 8 proof farms (distributed computational logic)

`http://skolemmachines.org/autolog/README.html`

22- ...

Thank You

23- afterwords . . .

Here is a link for a hint about the PROJECT on slide 18, re Kripke semantics:
`http://SkolemMachines.org/reports/BLAST18/kripke.pdf`