Constructive Coherent Translation of Propositional Logic

JRFisher@cpp.edu

(started: 2009, latest: May 26, 2017)

Abstract

Propositional theories are translated to coherent logic rules using what are called *fore-and-aft templates*. In order to illustrate the essential concepts this report also describes a prototype translator that is implemented using an ANTLR4 grammar, parser and its attendant tree-walking translator. The target of the translation is the *colog* language for coherent logic. Thus the design described here is one way to compile propositional logic problems for computation by a Skolem Machine [2]. The intention is to extend the fore and aft methods to a *constructive geometrization* of typed first-order logic using a *machine logic* framework.

The fore-and-aft translator is *constructive* because its use leads to translations of propositional theories or problems that are *sound* for intuitionistic logic. The question of intuitionistic *completeness* of the translation is problematic.

1 A Grammar for propositions

The following ANTLR 4 [1] grammar[1] is used to illustrate a constructive translator of propositional formulas into coherent logic. The coherent logic formulas that result are colog language rules for computation by a Skolem Machine.

[1] All embedded codes are `verbatim`.

```
/**
  *  Prop.g4
  *  An ANTLR 4 grammar for Propositions.
  *  2015.
  */
grammar Prop ;   // # labels for formula forms

// A theory is one or more assertions.
theory : statement+ ;

// A propositional statement is either
//    formula '.'    axiom
//    formula '?'    conjecture
//    formula '.?'   hypothetical
statement : f=formula end=('.' | '?' | '.?') ;

//////////
//   A formula has labelled cases, in order of precedence
//   N.B. position of <assoc=right>, vs what ANTLR4 book says
///////////////
formula :
     p=PROPOSITION                              # literal
   | NOT f=formula                              # negation
   | f1=formula IFF f2=formula                  # equivalence
   | f1=formula AND f2=formula                  # conjunction
   | f1=formula OR f2=formula                   # disjunction
   |<assoc=right> f1=formula IFTHEN f2=formula # implication
   | '(' f=formula ')'                          # parenthesized
   ;
   // Note all left-associative except '->'

// A proposition, the literal case for a formula
PROPOSITION : [a-z]+ [0-9]* ;

// The logical connectives lexed
NOT :     '~' ;
IFF :     '<=>' | '='  ;
AND :     '&' ;
OR :      '|' ;
IFTHEN :  '=>' ;
```

```
WS : [ \t\n\r]+ -> skip ; // ignore all whitespace
SL_COMMENT : '%' .*? '\n' -> skip ;  // single line
ML_COMMENT : '(*' .*? '*)' -> skip ;  // multiple line
```

The logical operators are all (automatically) left associative except implication which is declared as right associative. The tightness of the binding order of the logical operators is in accordance with the order of definition in the grammar code, negation `~` being most tightly binding, and implication `->` the least; otherwise parenthesize as usual. The `#` labels in the grammar are used to define listener methods associated with the generated parser.

For this translator `'=>'` is used of if-then and `'<=>'` and `'='` are taken as synonyms for if-and-only-if. Both `P<=>Q` and `P=Q` are replaced by `(P=>Q) & (Q=>P)` before fore-and-aft translation. In this regard, other translation designs would be possible.

2 Fore and Aft translation templates

A propositional *statement* (see the grammar) is a propositional formula followed by punctuation.

- axiom
  `P.`
- query
  `Q?`
- hypothetical
  `P => Q.?`

where P and Q are propositional formulas. A propositional *theory* (see the grammar) is a finite sequence of propositional statements. Informally, a theory is also called a *problem*.

The initial translation of an axiom to colog rule is
`true => '[P}'.`

The initial translation of a query is
`'{Q] => goal.`

And the initial translation of a hypothetical is

```
true => '[P}'.
'{Q] => goal.
```

A *fore form* '{P]' represents *evidence pending*, whereas an *aft form* '[P}' represents *evidence attained*. [2] Evidence pending refers to facts that may appear later on the branch of a Skolem machine computation, and evidence attained refers to facts that now appear on the branch.

The templates in the following table are used to guide the translation of subformulas in a recursive-descent fashion. One more detail of notation is required: In a fore form like '~{~P]', the formula context is ~P parsed in a subcontext which itself is negative (inherited from surrounding context). The reason that the fore-and-aft templates in the table are also called Heyting templates is explained in section 4.

Fore-and-Aft Heyting Templates

```
literal   p
   fore
      +
         p => '{p]'.
      ~
         ~p => '~{p]'.
      ~~
         p => '~~{p]'.
   aft
      +
         '[p}' => p.
      ~
         '~[p}' => ~p.
         ~p, p => false.
      ~~
         '~~[p}' => ~~p.
         ~p, ~~p => false.

negation  ~f
  fore
      +
```

[2] A previous LaTeXnotation used $\overleftarrow{P}$ and $\overrightarrow{P}$, respectively. The notation here is closer to that actually used by the translator and better represents negation subcontexts.

```
         '~{f]' => '{~f]'.
      ~
          '~~{f]' => '~{~f]'.
      ~~
          '~{f]' => '~~{~f]'.
   aft
      +
          '[~f}' => '~[f}'.
      ~
          '~[~f}' => '~~[f}'.
      ~~
          '~~[~f}' => '~[f}'.

conjunction f1&f2
   fore
      +
         '{f1]', '{f2]' => '{f1&f2]'.
      ~
         '~{f1]' => '~{f1&f2]'.
         '~{f2]' => '~{f1&f2]'.
         '~[f1&f2}' =>  => '~{f1&f2]'.
      ~~
         '{f1]', '{f2]' => '~~{f1&f2]'.
         '~~{f1]', '~~{f2]' => '~~{f1&f2]'.
   aft
      +
         '[f1&f2}' => '[f1}', '[f2}'.
      ~
         '{f1]', '{f2]', '~[f1&f2}' => false.
      ~~
         '~~[f1&f2}' => '~~[f1}'', ~~[f2}'.

disjunction f1|f2
   fore
      +
          '{f1]' => '{f1|f2]'.
          '{f2]' => '{f1|f2]'.
      ~
          '~{f1]', '~{f2]' => '~{f1|f2]'.
      ~~
```

```
          '~~{f1]' => '~~{f1|f2]'.
          '~~{f2]' => '~~{f1|f2]'.
          '~~[f1|f2}' => '~~{f1|f2]'.

   aft
      +
          '[f1|f2}' => '[f1} | [f2}'.
      ~
          '~[f1|f2}' => '~{f1]', '~{f2]'.
      ~~
          '~{f1]', '~{f2]', '~~[f1|f2}' => false.

implication  f1=>f2
   fore
      +
         '~{f1]' => '{f1=>f2]'.
         '{f2]' => '{f1=>f2]'.
         '[f1=>f2}' => '{f1=>f2]'.
      ~
         '{f1]', '~{f2]' => '~{f1=>f2]'.
         '~[f1=>f2}' => '~{f1=>f2]'.
      ~~
         '~~[f1=>f2}' => '~~{f1=>f2]'.
         '[f1=>f2}' => '~~{f1=>f2]'.
         '{f1=>f2]' => '~~{f1=>f2]'.
         '~{f1]' => '{f1=>f2]'.
         '{f2]' => '{f1=>f2]'.
         '~~{f2]' => '~~{f1=>f2]'.
   aft
      +
         '[f1=>f2}', '{f1]' => '[f2}'.
      ~
         '~[f1=>f2}' =>'~~[f1}', '~[f2}'.
         '{f1=>f2]', '~[f1=>f2}' => false.
         '~{f1]' => '{f1=>f2]'.
         '{f2]' => '{f1=>f2]'.
      ~~
         '~{f1=>f2]', '~~[f1=>f2}' => false.
         '{f1]', '~{f2]' => '~{f1=>f2]'.
```

The groups of templates (arranged by propositional connective) closely resemble the parse-tree walker code (`Prop2CL.java`) used to implement the translator: there are fore and aft cases, and cases for whether the context is positive, negative or double-negative. A convenient way to refer to a transform is by case. For example IA-3 would refer to the 3rd template of the - category for aft transforms for implication.

The templates are written in the form employed by the translator prototype. However, another notation that may be more pleasing is an over-arrow representation. For example, the IA+ template could be written as follows (from 2009 draft paper [10]):

$$\overrightarrow{f1 \Rightarrow f2},\ \overleftarrow{f1} \Rightarrow \overrightarrow{f2} \tag{1}$$

This notation is easier to write by-hand, looks tidier in print, but is not literally appropriate as output for the translator. $\overrightarrow{A}$ is `'[A}'` and $\overleftarrow{A}$ is `'{A]'`. Using this over-arrow style we list the following *classical-negation, or non-Heyting* transforms. These transforms are NOT employed by the translator.

$Classical - negation\ logic\ transforms$:

$\overrightarrow{\neg\neg\phi} \Rightarrow \phi$

$\overrightarrow{\neg(\phi \wedge \psi)} \Rightarrow \overrightarrow{\neg\phi} \vee \overrightarrow{\neg\psi}$

$\overrightarrow{\phi \Rightarrow \psi} \Rightarrow \overrightarrow{\neg\phi} \vee \overrightarrow{\psi}$

$\overrightarrow{\neg(\phi \Rightarrow \psi)} \Rightarrow \overrightarrow{\phi} \wedge \overrightarrow{\neg\psi}$

Here, ϕ and ψ are propositional subformulas. Using classical-negation transforms allows to use a simplified complete subset of AFT-ONLY transforms because classical negation of subformulas is recursively reductive (as specified in [2]).

Andrew Polonsky's thesis [9] provides a detailed description of a translator for first-order predicate logic using classical negation normal transforms.

3 Sample translations

A link to a translator prototype is posted in section 6. The following translations were automatically generated by PropTool.

Example 1. Consider the propositional theory

```
p & q.          % axiom
p => r | s.     % axiom
q => r | t.     % axiom
r | s | t?      % query
```

The translator generates the following verbose translation.

```
% axiom: p&q.
   true => '[p&q}'.
   '[p&q}' => '[p}', '[q}'.
   '[p}' => p.
   '[q}' => q.
% axiom: p->r|s.
   true => '[p=>r|s}'.
   '[p=>r|s}', '{p]' => '[r|s}'.
   p => '{p]'.
   '[r|s}' => '[r}' | '[s}'.
   '[r}' => r.
   '[s}' => s.
% axiom: q->r|t.
   true => '[q=>r|t}'.
   '[q=>r|t}', '{q]' => '[r|t}'.
   q => '{q]'.
   '[r|t}' => '[r}' | '[t}'.
   '[r}' => r.
   '[t}' => t.
% conjecture: r|s|t?
   '{r|s|t]' => goal.
   '{r|s]' => '{r|s|t]'.
   '{t]' => '{r|s|t]'.
   '{r]' => '{r|s]'.
   '{s]' => '{r|s]'.
   r => '{r]'.
   s => '{s]'.
   t => '{t]'.
```

or the following terse, folded translation.

```
% axiom: p&q.
   true => p, q.
% axiom: p=>r|s.
```

```
    p => '[r|s}'.
   '[r|s}' => r | s.
% axiom: q=>r|t.
    q => '[r|t}'.
   '[r|t}' => r | t.
% conjecture: r|s|t?
   r => goal.
   s => goal.
   t => goal.
```

Example 2. Consider the propositional theory

```
~(a | ~b).     % axiom
~(a | ~b)?     % query
```

The translator generates the following verbose translation. This example illustrates the interconnection between fore and aft forms when the *aft reduction of negation* reduces all the way to literals.

```
% consistency
   ~a, a => false.
   ~b, ~~b => false.
% axiom: ~(a|~b).
   true => '[~(a|~b)}'.
   '[~(a|~b)}' => '~[(a|~b)}'.
   '~[(a|~b)}' => '~[a|~b}'.
   '~[a|~b}' => '~[a}', '~[~b}'.
   '~[a}' => ~a.
   '~[~b}' => '~~[b}'.
   '~~[b}' => ~~b.
% conjecture: ~(a|~b)?
   '{~(a|~b)]' => goal.
   '~{(a|~b)]' => '{~(a|~b)]'.
   '~{a|~b]' => '~{(a|~b)]'.
   '~{a]', '~{~b]' => '~{a|~b]'.
   ~a => '~{a]'.
   '~~{b]' => '~{~b]'.
   ~~b => '~~{b]'.
   b => ~~b.
```

and the terse version is

```
% consistency
   ~a, a => false.
% axiom: ~(a|~b).
   true => ~a, ~~b.
% conjecture: ~(a|~b)?
   ~a, ~~b => goal.
```

Example 3. Consider the propositional problem

```
p & q.
p => b & a | r.
q => ~(a & b).
r?
```

The translator generates the following terse translation. (Do the verbose translation and folding by hand as an exercise.) This example illustrates the interconnection between fore and aft forms when the *aft reduction of negation* does NOT reduce all the way to literals: A consistency rule is introduced, a reversal rule is introduced for negation, and the corresponding fore reduction of negation does reduce all the way to literals. Irreducible aft negations need to be exposed to a consistency check!

```
% consistency
   ~b, b => false.
   ~a, a => false.
   % '{a&b]', '~[a&b}' => false.
   a, b,'~[a&b}' => false.
% axiom: p&q.
   true => p, q.
% axiom: p=>b&a|r.
    p => b, a | r.
% axiom: q=>~(a&b).
    q => '~[a&b}'.
    % reversal
    '~[a&b}' => '~{a&b]'.
   ~a => '~{a&b]'.
   ~b => '~{a&b]'.
% conjecture: r?
   r => goal.
```

The Skolem Machine proof in Figure 1 was automatically generated for the terse colog theory. For this problem, the proof depends on the consistency check, but does does not require the aft-to-fore reversal.

Example 4. Consider the propositional problem

```
~(a & b).
~(a & b)?
```

Translate the problem and show that a proof for this problem requires the aft-to-fore reversal but not the consistency check. Also show that a similar observation applies to the tautology problem ~`(a & b) => ~(a & b).?`

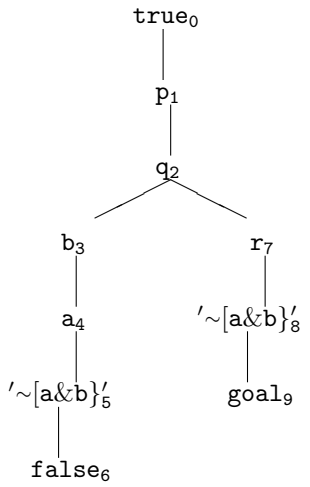

Figure 1: Example 3 Proof

There is some irony that a *perfect* folding algotithm might simply produce a rule like `true => goal` . However, this would remove all the harmony of the fore-and-aft translation process.

4 Intuitionistic soundness of fore-and-aft translation

It can be shown that each template follows a pattern for some corresponding intuitionistically valid Heyting algebra formula [11]. The aft template `‘[P => Q}’, ‘{P]’ => ‘[Q}’` is a good example, and the valid Heyting algebra formula could be written $(P \Rightarrow Q) \wedge P \Rightarrow Q$. That is, given that $(P \Rightarrow Q)$ is an axiom (presumed evidence) and then that P becomes evident, one can conclude Q.

Each template can be intuitively interpreted as a generalized statement about evidence that appears on a branch of an active Skolem Machine and

how new evidence should be asserted to the branch. This approach gives a kind of dynamical machine *operational semantics* based on the *colog-tree* theory of evidence presented in [7]. However, this interesting aspect can be summarized by saying that the templates correspond to intuitionistically valid Heyting algebra formulas.

Exercise A. Compare the operational semantics using translation instances of the Heyting templates on a branch of a Skolem Machine to natural deduction and the sequent calculus. See, e.g., [8], section 1.3.

Exercise B. Use Coq [3] to prove that the Heyting transforms are intuitionistically valid. Also, formulate the Heyting transforms as queries for algebraic logic theories and then use Prover9 [4] to prove that the transforms are intuitionistically valid.

Any propositional axiom is asserted as a fact to a branch and any such fact can be transformed using valid intuitionistic formulas (the results of using the templates) to give new facts. Thus the colog theory resulting from the translation of all of the axioms is a constructive logical consequence of the propositional axioms. Similarly, the translations of queries and hypotheticals are constructive consequents of the corresponding propositions. That is, if Q were a consequence axioms $\{A_i\}$ then `Q => goal` would be also a satisfiable coherent rule, and similarly for a hypothetical (but see proviso given below).

Perhaps the easy way to check the intuitionistic validity of a Heyting algebra formulas corresponding to templates is to use the Heyting algebra of open subsets of the real line. So, for example, the formula $(P \to Q) \wedge P \to Q$ would correspond to $int(P^c \cup Q) \cap P \subseteq Q$, and the latter formula can be proved using open-set arguments for the real line with familiar topology.

Soundness does require a restriction on the use of *hypotheticals*, $P \to Q$.? A simple example of this would be the following little problem:

```
p => q.?
q => p.?
```

The translated theory would be :

```
true => p,
q => goal.
true => q.
p => goal.
```

Notice that the hypothetical assumption of each provides evidence for the conclusion of the other hypothetical, thus allowing unsound proofs. Thus, we should only allow one hypothetical in a given propositional problem, or check that two or more hypotheticals do not mix assertions with goals, something which is not automated in the translator. (That issue is decidable, but it is very awkward to impose this condition on a language translator.)

5 Completeness issues

What one might desire for completeness of Heyting transform translation is that, if a propositional problem has any constructive proof decision using established methods, then there is a suitably constructed fore-and-aft translation of the problem to colog that has a Skolem Machine proof. [3]

The efforts described in this report is a work-in-progress which falls short of general completeness. However, various short positions appear to be interesting.

The qualification required regarding hypotheticals given in the previous section is related to a somewhat *stickier* problem related to completeness of the translator. For, example $p \wedge q \Rightarrow p$ is valid but the translator translates `p & q => p?` too conservatively, whereas a hypothetical `p & q => p.?` does suffice, or, one can reformulate as an axiom and a query `p & q.  p?`. Another good example of this is the intuitionistic tautology `~a|~b => ~(a & b).?`

A similar ackwardness involves interchangeability of $\neg P$ and $P \Rightarrow false$, especially in a goal. For example, one might hope that query `b => ~a` should follow from axiom `a,b -> false` or the axiom `~(a & b)` in direct fashion. However, at this writing, this problem cannot be directly formulated in that manner. It is often possible to give a simple alternate problem formulation that might suffice.

There is a more serious shortcoming with regard to the nonreductive aft transformations such as CA-. (Such transforms are not completely recursively reductive via the connective – they lack the *classical harmony*). So, for example, the problem `~(a & a).  ~a?` will not automatically prove because there is no way to effectively express that `a&a=a` (in the present translator approach). Or, more generally there is currently not effective way to express that if `P=Q` then `~P=~Q` where P and Q are propositional variables.

Here are some questions that might lead to good projects ...

[3] For an excellent historical survey of known decision procedures, see Dychoff [6].

1 What is the scope or limitation of what can be proved using the current templates, including problem formulation tactics? Is the current translation approach effectively complete for formulas only having negation in front of literals (and not subformulas)?

2 Is it possible to write effective *scripts/tactics* for problems, whereby a script describes both the logic problem and a suggested approach to search for a proof of the problem?

Perhaps it is cogent to mention that fore-and-aft translation does not produce an *extension* of the input propositional theory, strictly speaking. What is produced here is more like a *compilation/translation* of the input to machine code. As such, the approach here might be considered as an experiment in *applied proof semantics*. Reference [5] focuses on *coherent conservative extension* of FOL, but extension concepts and replacement concepts seem very much intertwined, pending more study, and [5] effectively discusses this aspect also.

6 Download PropTool and sources

A demonstration translator is available as a self-starting jar file:
`http://SkolemMachines.org/reports/prop2cl/PropTool.jar`
The Java Sources are included in the jar. One might need to allow the jar to execute – open with Jar Launcher. The jar-cautious reader can extract the jar first to check that it is benign. Ocassionally, a newer version of PropTool may translate a problem a little differently than characterized in Section 3 above. Also, terse translations are not always as terse as might be possible.

References

[1] Antlr grammar tools: `http://www.antlr.org`

[2] M. Bezem and T. Coquand, Automating Coherent Logic. In G. Sutcliffe and A. Voronkov, editors, *Proc. LPAR-12*, LNCS 3825, 2005, pages 246-260.

[3] Coq `https://coq.inria.fr`

[4] Prover9 `https://www.cs.unm.edu/~mccune/mace4/`

[5] R. Dyckhoff and S. Negri, Geometrisation of first-order logic, *Bulletin of Symbolic Logic* 21, pp 123–163, 2015. (http://rd.host.cs.st-andrews.ac.uk/publications/Geometrisation-FOL.pdf)

[6] Roy Dychoff, Intuitionistic decision procedures since Gentzen, October 8, 2014, preprint.

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

[8] Sara Negri and Jan Von Plato, *Structural Proof Theory*, Cambridge, 2001.

[9] Andrew Polonsky, *Proofs, Types, and Lambda Calculus*, doctoral thesis, UiB, December 4, 2010.

[10] Earlier (2009) notes regarding fore-and-aft translation: Linked at SkolemMachines website: `fol2cl.pdf`

[11] `http://en.wikipedia.org/wiki/Heyting_algebra`

[12] `https://en.wikipedia.org/wiki/Intermediate_logic`