Logic + Control: An example
or
SAT solver of Howe & King as a logic program

(File `./LIPIcs/29.pdf`)

Włodzimierz Drabent

Institute of Computer Science, Polish Academy of Sciences
Linköping University (Sweden)

http://www.ipipan.waw.pl/~drabent

ICLP'12, 6th September 2012
Version compiled on September 10, 2012

This file contains extra material, not intended to be shown within a short presentation. In particular, such are all the slides with their titles in parentheses.

Is there logic / "logic" in actual Logic Programming ?

To which extent LP is declarative/logical ?

How to reason about logic programs?

We present

a construction of a practical Prolog program
(SAT solver of Howe&King).

Most of the reasoning done at the declarative level
(formally)
abstracting from any operational semantics.

Plan

- Specification
- Proving correctness & completeness
- Logic programs 1, 2, 3
- Adding control
- Conclusions

Preliminaries

Definite programs.

To describe relations to be defined by program predicates:

Specification – a Herbrand interpretation S.

Specified atom – a $p(t_1, \ldots, t_n) \in S$.

Specifying a SAT solver

So apparently
a SAT solver should compute L_2^0.

Computing exact L_2^0 unnecessary.

E.g. nobody uses append/3 defining the list appending relation exactly!

☞ Common in LP: relations to be computed known approximately.

Representation of propositional formulae

for a SAT solver [Howe&King]

Literals	x	$\neg x$
as pairs	`true-X`	`false-X`

CNF formulae $(\ldots \wedge (\ldots \vee Literal_{ij} \vee \ldots) \wedge \ldots)$
as lists of lists $[\ldots,\ [\ldots,\ Pair_{ij},\ \ldots],\ \ldots]$

CNF formula $[f_1, \ldots, f_n]$ is satisfiable iff
it has an instance $[f_1\theta, \ldots, f_n\theta]$ where $\forall_i$
$f_i\theta \in L_1^0 = \{\, [t_1\text{-}u_1, \ldots, u\text{-}u, \ldots, t_n\text{-}u_n] \in \mathcal{H} \,\}$.

CNF formula f is satisfiable iff
some $f\theta$ is in $L_2^0 = \{\, [f_1\theta, \ldots, f_n\theta] \mid \text{as above} \,\}$.

A program defining L_2^0 is a SAT solver.

Specifying a SAT solver

So
a SAT solver ~~should~~ may compute L_2^0.

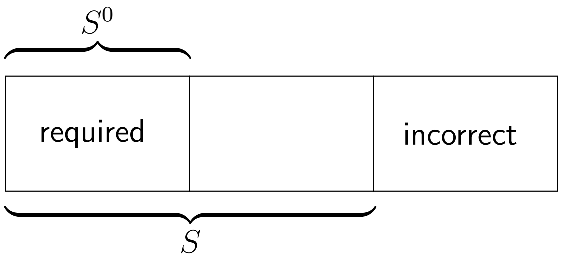
Also
it may compute a certain $L_2 \supseteq L_2^0$.

$L_2 = \{\, s \in \mathcal{H} \mid \text{ if } s \text{ is a list of lists of pairs then } s \in L_2^0 \,\}$.

Any set $L_2^0 \subseteq L_2' \subseteq L_2$ will do:
a CNF formula f is satisfiable iff some $f\theta$ is in L_2'.

☞ Common in LP: relations to be computed known approximately.

Approximate specifications



Approximate specification – (S^0, S), where $S^0 \subseteq S$.
↑ for completeness ↑ for correctness

Intention: $S^0 \subseteq M_P \subseteq S$. S^0 – what has to be computed.
S – what may be computed.

(Details – 1st specification for SAT solver)

Specification: (S_1^0, S_1) with the specified atoms

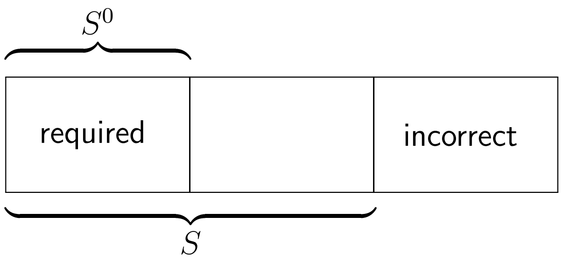
S_1^0:
$sat\_cnf(t)$, where $t \in L_2^0$,
$sat\_cl(s)$, $s \in L_1^0$,
$x = x$, $x \in \mathcal{H}$

S_1:
$sat\_cnf(t)$, where $t \in L_2$,
$sat\_cl(s)$, $s \in L_1$,
$x = x$, $x \in \mathcal{H}$

$$L_1^0 = \{\, [t_1\text{-}u_1, \ldots, u\text{-}u, \ldots, t_n\text{-}u_n] \in \mathcal{H} \,\},$$
$$L_2^0 = \{\, [s_1, \ldots, s_n] \mid s_1, \ldots, s_n \in L_1^0 \,\},$$

$$L_1 = \{\, t \in \mathcal{H} \mid \text{if } t \text{ is a list of pairs then } t \in L_1^0 \,\},$$
$$L_2 = \{\, s \in \mathcal{H} \mid \text{if } s \text{ is a list of lists of pairs then } s \in L_2^0 \,\}.$$

Approximate specifications



Approximate specification for SAT solver: (S_1^0, S_1),

states that predicate sat_cnf defines a set L_2': $L_2^0 \subseteq L_2' \subseteq L_2$.

[Details ⇝ the paper]

Correctness & completeness of programs

Correctness (imperative programming)
↙ ↘
Correctness $M_P \subseteq S$ | Completeness $S \subseteq M_P$ (logic programming)

Completeness:
Everything required by the spec. is computed.

Correctness:
Everything computed is compatible with the spec.

P semi-complete w.r.t. S
= P complete for terminating queries
(under *some* selection rule).

[Details ⇝ the paper]

Correctness & completeness, sufficient conditions

Th. (Clark 1979): P **correct** w.r.t. S when
for each $(H \leftarrow B) \in ground(P)$, $B \subseteq S \Rightarrow H \in S$.

(Out of correct atoms, the clauses produce only correct atoms.)

Th.: P **semi-complete** w.r.t. S when
for each $H \in S$,
exists $(H \leftarrow B) \in ground(P)$ where $B \subseteq S$.

(Each required atom can be produced out of required atoms.)

Semi-complete + terminating $\Rightarrow$ complete.

SAT solver 1

P_1:
$sat\_cnf([\,])$.
$sat\_cnf([Clause|Clauses]) \leftarrow sat\_cl(Clause),\ sat\_cnf(Clauses)$.
$sat\_cl([Pol\text{-}Var|Pairs]) \leftarrow Pol = Var$.
$sat\_cl([H|Pairs]) \leftarrow sat\_cl(Pairs)$.

Can be constructed
guided by the sufficient conditions above, and specification (S_1^0, S_1).

Correct w.r.t. S_1, complete w.r.t. S_1^0. [Details ⇝ the paper]

Inefficient backtracking search.

(Towards better efficiency)

Idea: Watch two variables of each clause.
Delay $Pol = Var$ in $sat\_cl([Pol\text{-}Var|Pairs]) \leftarrow Pol{=}Var$
until Var watched and bound.

New predicates – another representations of clauses
E.g. (v_1, p_1, v_2, p_2, s) for $[p_1\text{-}v_1, p_2\text{-}v_2|s]$.
To block on v_1, v_2
Specification (S_1^0, S_1) extended ⇝ (S_2^0, S_2).

Guided by the sufficient conditions for correctness & completeness
a logic program P_2 built,
correct & complete w.r.t. the new specification.
[Details ⇝ the paper]

(Towards efficiency. Details: the new spec.)

Idea: Watch two variables of each clause.
delay $Pol = Var$ in $sat\_cl([Pol\text{-}Var|Pairs]) \leftarrow Pol{=}Var$
until Var watched and bound.

New predicates. Specification: S_1^0 (resp. S_1) extended by atoms

$sat\_cl3(s, v, p)$, where $[p\text{-}v|s] \in L_1^0$ (resp. $\in L_1$),
$sat\_cl5(v_1, p_1, v_2, p_2, s)$,
$sat\_cl5a(v_1, p_1, v_2, p_2, s)$, $[p_1\text{-}v_1, p_2\text{-}v_2|s] \in L_1^0$ (resp. $\in L_1$).
Already in S_1^0 (S_1):
$sat\_cl(s)$ $s \in L_1^0$ (resp. $\in L_1$).

Intention: v_1, v_2 – the watched variables
`:-block sat_cl5(-,?,-,?,?)`
sat_cl5a called with v_1 bound

(Towards efficiency, final logic program)

P_2 may flounder (under the intended delays).

To avoid floundering – new predicates, new specification.

Initial queries $sat(f, l)$
$\uparrow$
Variables in f

Spec. requires l to be a list of $true/false$

Guided by the sufficient conditions for correctness & completeness
a logic program $P_3 \supseteq P_2$, correct & complete.
[Details ⇝ the paper]

Towards better efficiency – brief

To prepare the intended control – new predicates.

E.g. another data representation, like
(v_1, p_1, v_2, p_2, s) for $[p_1\text{-}v_1, p_2\text{-}v_2|s]$,
to block on v_1, v_2.

Specification (S_1^0, S_1) extended ⇝ (S_3^0, S_3).

Guided by the sufficient conditions for correctness & completeness
a logic program P_3 built
correct & complete w.r.t. the new specification.

Adding control to P_3

- Delays – modifying the selection rule
  `:-block sat_cl5(-,?,-,?,?)`
- Two cases of pruning SLD-trees.
  Skipping a rule of P_3; implemented by `( ...-> ...;... )`.

  Completeness preserved.
  Case 1 – proof [technical report].
  Case 2 – informal justification

Result: Prolog program [Howe&King] of 22 lines / 12 rules.
Implements DPLL with watched literals and unit propagation.
(partly)

(Adding control, details)

Delays – modifying the selection rule
`:-block sat_cl5(-,?,-,?,?)`

Pruning 1. Choosing one of two clauses dynamically.
Completeness preserved. [Proof → tech. report]

$sat\_cl5(Var1, \ldots, Var2, \ldots) \leftarrow sat\_cl5a(Var1, \ldots, Var2, \ldots).$
$sat\_cl5(Var1, \ldots, Var2, \ldots) \leftarrow sat\_cl5a(Var2, \ldots, Var1, \ldots).$

⇝

$$sat\_cl5(Var1, \ldots, Var2, \ldots) \leftarrow$$
$$nonvar(Var1) \rightarrow sat\_cl5a(Var1, \ldots, Var2, \ldots)$$
$$;\ sat\_cl5a(Var2, \ldots, Var1, \ldots).$$

(Adding control, details 2)

Pruning 2. Removing a redundant part of SLD-tree.
(Do not work on a clause which is already true.)
Completeness preserved, informal justification.

$$sat\_cl5a(Var1, Pol1, \_, \_, \_) \leftarrow Var1 = Pol1.$$
$$sat\_cl5a(\_, \_, Var2, Pol2, Pairs) \leftarrow sat\_cl3(Pairs, Var2, Pol2).$$
$$\wr$$
$$sat\_cl5a(Var1, Pol1, Var2, Pol2, Pairs) \leftarrow$$
$$Var1 = Pol1 \ \rightarrow \ true; \quad sat\_cl3(Pairs, Var2, Pol2).$$

Conclusions, proving correctness & completeness

Proving correctness.
Method of [Clark'79] simpler / not weaker than that of Bossi&Cocco [Apt'97].
☹ Neglected.

Proving completeness. Seldom considered. (E.g. not in [Apt'97].)
Our method: new notion of semi-completeness,
semi-completeness + termination ⇒ completeness.

Both methods
- ☺ simple, natural, declarative (but termination),
- ☺ correspond to common-sense reasoning about programs,
- ☺ applicable in practice (maybe informally).

Ex.: An error in P_1 (first version) found & located by a failed proof attempt.

Methods for programs with negation: [Drabent,Miłkowska'05]

Conclusions, approximate specifications

➔ Approximate spec's crucial for formal / precise reasoning about programs.

Exact relations (defined by programs) often not known,
not easy to understand.

Ex.: Which set is defined by $sat\_cl/1$ in P_1? In P_2, P_3?
Misunderstood by the author (first report) and some reviewers.

➔ Approximate spec's useful for declarative diagnosis (DD).
Trouble: DD requires exact specifications.
Ex. Is $append([a], b, [a|b])$ correct?

Approximate spec's should be used:

Diagnosing incorrectness / incompleteness – specification for correctness / completeness

Conclusions, approximate specifications 2

Transformational approaches seem inapplicable
to our example $P_1 \rightsquigarrow P_3$,

as the same predicates define different sets in P_1, P_3.
have the same approximate specification

Interpretations as specifications
– "existential specifications" inexpressible. ☹

Ex.: We could not state that
for each satisfiable f some true instance $f\theta$ is computed.
We required all true instances.

Solution(?): Use *theories* as specifications.

Conclusions, declarative programming

Most of reasoning can be done
at declarative level / pure logic programs.

Abstracting from operational semantics,
thinking in terms of relations;
formally.

Separation "logic" – "control" works:

Reasoning related to operational semantics / efficiency
independent from that related to correctness & semi-completeness.

But: Pruning may spoil completeness.

Conclusions, ...

Claim: The presented approach can be used in practice,
maybe informally,
in programming and in teaching.

LP is not declarative unless
we have/use declarative means of reasoning about programs.

Conclusions, summary

- Approximate specifications crucial
  Approximate spec's useful for declarative diagnosis
- Simple methods for proving correctness & completeness
  declarative (but termination)
  applicable in practice
- Most of reasoning can be done at declarative level
  (pure logic programs)
  Declarative properties
  Operational properties – reasoning independent
- Claim: Approach practically applicable maybe informally,
  in programming and in teaching.

http://www.ipipan.waw.pl/~drabent