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

Is there 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

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.
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 \( L^0_2 \).

Computing exact \( L^0_2 \) 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**
  - \( x \)
  - \( \neg x \)

- **true-X**
- **false-X**

- **CNF formulae**
  - \( (\ldots \land (\ldots \lor \text{Literal}_{ij} \lor \ldots) \land \ldots) \)

- **as lists of lists**
  - \( \ldots, \ldots, [\text{Pair}_{ij}, \ldots], \ldots \)

- **CNF formula** \( [f_1, \ldots, f_n] \) is satisfiable iff
  - it has an instance \( [f_1\theta, \ldots, f_n\theta] \) where \( \forall i f_i\theta \in L^0_2 = \{ [t_1-u_1, \ldots, u-u, \ldots, t_n-u_n] \in \mathcal{H} \} \).

- **CNF formula** \( f \) is satisfiable iff
  - some \( f\theta \) is in \( L^0_2 = \{ [f_1\theta, \ldots, f_n\theta] \} \) as above.

A program defining \( L^0_2 \) is a SAT solver.

Common in LP: relations to be computed known approximately.
Approximate specifications

Approximate specification - $(S^0, S)$, where $S^0 \subseteq S$.

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

Approximate specifications

Approximate specification for SAT solver: $(S^0_1, S_1)$, states that predicate $\text{sat.cnf}$ defines a set $L'_2$: $L^0_2 \subseteq L'_2 \subseteq L_2$.

[Details -> the paper]

Correctness & completeness of programs

Correctness (imperative programming)

$M_P \subseteq S$

Completeness (logic programming)

$S \subseteq M_P$

Correctness:

Everything required by the spec. is computed.

Completeness:

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 \( (H \leftarrow B) \in \text{ground}(P) \), \( B \subseteq S \Rightarrow H \in S \).

(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 \( (H \leftarrow B) \in \text{ground}(P) \) where \( B \subseteq S \).

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

Semi-complete + terminating \( \Rightarrow \) complete.

---

**(Towards better efficiency)**

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

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

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

---

**(Towards efficiency. Details: the new spec.**)

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

New predicates. Specification: \( S_1^0 \) (resp. \( S_1 \)) extended by atoms
\[
\text{sat}\_cl3(s, v, p), \quad \text{where } [p-v|s] \in L_1^0 \text{ (resp. } \in L_1),
\]
\[
\text{sat}\_cl5(v_1, p_1, v_2, p_2, s), \quad [p_1-v_1, p_2-v_2|s] \in L_1^0 \text{ (resp. } \in L_1).
\]
Already in \( S_1^0 \) (\( S_1 \)):
\[
\text{sat}\_cl(s), \quad s \in L_1^0 \text{ (resp. } \in L_1).
\]

Intention: \( v_1, v_2 \) – the watched variables

\( :-\text{block sat}\_cl5(-,?,-,?,?) \)
\( \text{sat}\_cl5a \) 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 $\text{sat}(f, l)$
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 $[p_1-v_1, p_2-v_2][s]$, to block on $v_1, v_2$.

Specification $(S_0^1, S_1)$ extended $\rightsquigarrow (S_0^3, S_3)$.

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
  $:-\text{block sat_cl5}(-,-,?,?,?)$

- Two cases of pruning SLD-trees.
  Skipping a rule of $P_3$; implemented by $(\ldots\rightarrow\ldots;\ldots)$.
  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
$:-\text{block sat_cl5}(-,-,?,?,?)$

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

\[
\begin{aligned}
\text{sat_cl5}(Var_1,\ldots, Var_2,\ldots) & \leftarrow \text{sat_cl5a}(Var_1,\ldots, Var_2,\ldots). \\
\text{sat_cl5}(Var_1,\ldots, Var_2,\ldots) & \leftarrow \text{sat_cl5a}(Var_2,\ldots, Var_1,\ldots).
\end{aligned}
\]

\[
\begin{aligned}
\text{sat_cl5}(Var_1,\ldots, Var_2,\ldots) & \leftarrow \\
\text{nonvar}(Var_1) & \rightarrow \text{sat_cl5a}(Var_1,\ldots, Var_2,\ldots) \\
& ; \text{sat_cl5a}(Var_2,\ldots, Var_1,\ldots).
\end{aligned}
\]
(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.

\[
\begin{align*}
\text{sat}_\text{cl5a}(\text{Var1}, \text{Pol1}, \ldots) & \leftarrow \text{Var1} = \text{Pol1}, \\
\text{sat}_\text{cl5a}(\ldots, \text{Var2}, \text{Pol2}, \text{Pairs}) & \leftarrow \text{sat}_\text{cl3}(\text{Pairs}, \text{Var2}, \text{Pol2}). \\
\end{align*}
\]

\[
\begin{align*}
\text{sat}_\text{cl5a}(\text{Var1}, \text{Pol1}, \text{Var2}, \text{Pol2}, \text{Pairs}) & \leftarrow \\
\text{Var1} = \text{Pol1} \rightarrow \text{true}; \\
\text{sat}_\text{cl3}(\text{Pairs}, \text{Var2}, \text{Pol2}). \\
\end{align*}
\]

**Conclusions, approximate specifications**

- **Approximate spec's crucial for formal reasoning about programs.**
  - Exact relations (defined by programs) often not known, not easy to understand.
  - **Ex.** Which set is defined by \text{sat}_\text{cl}/1 in \text{P}_1? In \text{P}_2, \text{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 \text{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 \text{P}_1 \rightsquigarrow \text{P}_3,
  - as the same predicates define different sets in \text{P}_1, \text{P}_3.
  - have the same approximate specification

- Interpretations as specifications
  - "existential specifications" inexpressible.

- **Ex.:** We could not state that
  - for each satisfiable \text{f some} true instance \text{fθ} is computed.
  - We required all true instances.

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

---

**Ex.:** An error in \text{P}_1 (first version) found & located by a failed proof attempt.

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