It Is Declarative
On Reasoning about Logic Programs

Włodzimierz Drabent
IPI PAN, Oriona 21, Pl – 01-237 Warszawa, Poland, and
IDA, Linköpings universitet, S – 581 83 Linköping, Sweden
wdr@ida.liu.se, http://www.ipipan.waw.pl/~drabent

The subsequent pages contain (most of) the poster presented at the 1999
International Conference on Logic Programming.

Page 13 should contain the definition of partial correctness proof method from
[Apt97, Chapter 8]. Some lines on p. 4 and some arrows describing data flow in
the example program of p. 16 are missing.

Abstract

We advocate using the declarative reading of logic programs in proving
partial correctness, when the properties of interest are declarative. Some
publications present unnecessarily complicated methods for proving such
properties. These approaches refer to the operational semantics, as they
consider calls and successes of the predicates of the program during LD-
resolution. We show that this is an unnecessary complication and that a
straightforward proof method is simpler and in some sense more general.
Our approach is based solely on the property that “whatever is computed
is a logical consequence of the program”. This approach is not new and
can be traced back to the work of Clark in 1979, However it seems that it
has been - to a certain extent - forgotten. We believe in its importance in
teaching logic programming.

We complement the abovementioned method of proving (partial) cor-
rectness by a method of proving completeness. We point out that usually
one is interested in soundness and correctness w.r.t. two different specifi-
cations. The specification for correctness states which atoms are allowed to
be computed, the other which have to be computed. For instance for a non
list z, atom append([], z, z) is neither considered incorrect nor required to
be computed.

References


[DM88] W. Drabent and J. Małuszynski. Inductive Assertion Method for Logic Pro-


It Is Declarative

On reasoning about logic programs

Wlodzimierz Drabent

Polish Academy of Sciences

&

Linköping University

www.ipipan.waw.pl/~drabent

Introduction

An apparent belief:

To prove properties of logic programs one has to reason in terms of the operational semantics (modes / the form of procedure calls / selection rule / ...).

E.g.: [Apt97,Chapter8], [Pedreschi,Ruggieri99], ...

But this implies:

Logic Programming is not declarative.

We disprove this belief!

Preliminaries

Programs: Horn clause programs.

Answers: answer instances of queries.

Q is an answer iff P |= Q
(by soundness and completeness of SLD-resolution).

Specifications

Different specifications for correctness and completeness!

Correctness & completeness are independent from the operational semantics (in particular from call patterns).
Example. Specification of append:

1. for correctness

\[ spec = \left\{ \text{app}(k, l, m) \in \mathcal{H} \mid \begin{array}{l}
\text{if } l \text{ or } m \text{ is a list then } \\
k, l, m \text{ are lists} \\
\text{and } k \cdot l = m
\end{array} \right\} \]

(\mathcal{H} - \text{Herbrand universe,} \\
\ast - \text{list concatenation.})

2. for completeness

\[ spec' = \left\{ \text{app}(k, l, m) \in \mathcal{H} \mid k, l, m \text{ are lists} \\
\text{and } k \cdot l = m \right\} \]

Declarative Proof Method for Correctness

([Clark 79] or earlier)

\[ P - \text{Horn clause program} \]
\[ Q - \text{computed instance of a query} \]

\[ SPEC \models Q \text{ if } SPEC \models P \]

(To prove a program correct, show \( SPEC \models C \) for each its clause \( C \).)

Sound, as \( P \models Q \). (This is the whole proof!)

Complete (see e.g. [Deransart 93]).

[proving correctness]

Example

Program:
\[ \text{app}([], L, L) \]
\[ \text{app}([H|K], L, [H|M]) \leftarrow \text{app}(K, L, M) \]

Specification: \( spec \) from the previous example.

We have to show

\[ spec \models \text{app}([], L, L) \quad (1) \]
\[ spec \models \text{app}([H|K], L, [H|M]) \leftarrow \text{app}(K, L, M) \quad (2) \]

Proof of (2): Take values \( h, k, l, m \) for \( H, K, L, M \) such that \( spec \models \text{app}(k, l, m) \). If \( l \) or \( [h|m] \) is a list then \( l \) or \( m \) is a list, then \( k, l, m \) are lists and \( k \cdot l = m \).

Thus \( [h|k] \cdot l = [h|m] \). So \( spec \models \text{app}([h|k], L, [h|m]) \).

The method is declarative

program: computed instance: implications
specification: interpretation / theory
correctness: \( spec \models Q \)
proof: showing \( spec \models P \)

No reference to computation, SLD, LD, selected atom, query, ...

For specifications being theories, a typical spec-
ification of a predicate is:

\[ p(x) \leftrightarrow \bigwedge_i \varphi_i \rightarrow \psi_i \]

To prove: implications of the form

\[ (\bigwedge_i \varphi^k_j \rightarrow \psi^k_i) \rightarrow (\varphi_i \rightarrow \psi_i) \]
Proving completeness

Technicalities

1. ONLY-IF($P$)
   - Program $P$ with implications reversed.

   For each predicate symbol $p$, if $P$ contains
   
   $p(x_1) \leftarrow B_1$
   $\cdots$
   $p(x_k) \leftarrow B_k$

   then ONLY-IF($P$) contains

   $p(\bar{x}) \rightarrow \bigvee_{i=1}^{k} \exists \bar{\alpha} \bar{x} = \bar{\alpha} \land B_i$.

2. $\textit{spec eq} := \{t = t \mid t \text{ is a ground term}\}$.

Theorem: $P$ a program, $Q$ a query. If

(i) $\textit{spec}\_2 \cup \textit{spec eq} \models \text{ONLY-IF}(P)$,

(ii) $P$ terminates for $Q$, i.e. there exists a finite SLD-tree for $Q$ and $P$.

Then

1. if $\textit{spec}\_2 \models \exists Q$ then $P \models \exists Q$ (and $Q$ succeeds with $P$).

2. if $\textit{spec}\_2 \models Q$ then $P \models Q$ (and $Q$ is an answer of $P$).

Example. Program APPEND:

app([],L,L)
app([H|K],L,[H|K]) :- app(K,L,L)

ONLY-IF(APPEND):

$app(x,y,z) \rightarrow$

$\rightarrow x = [], y = z \lor$

$\exists h,k,l,m : x = [h|k], y = l, z = [h|m], app(k,l,m)$

$\textit{spec}\_2 = \{ app(k,l,m) \in \mathcal{H} \mid k,l,m \text{ are lists and } k * l = m \}$

We have $\textit{cspec}\_2 \cup \textit{spec eq} \models \text{ONLY-IF}(\text{APPEND})$

Let $Q = app(x,y,m)$, where $m$ is a list.
It can be proved that $P$, $Q$ terminate.

$\textit{spec}\_2 \models \exists Q$, thus $Q$ succeeds.

Let $k,l$ be lists such that $k * l = m$; $Q' = app(k,l,m)$.
Then $\textit{spec}\_2 \models Q'$. Thus $Q'$ is an answer of $P$.
Hence $Q'$ (or a more general atom) is an answer for $P,Q$.

So as the results we obtain all the splittings of list $m$. 
Operational proof method
for correctness

[Apt97, PR99] use the method of [Bossi,Cocco89],
which is an instance of [Drabent,Matuszyński88].

It considers procedure calls & successes in LD-resolution.

Specifications: pre- & postconditions

Correctness: calls ⊆ precondition,
successes ⊆ postcondition

The verification condition:
  one implication per atom of \( P \).
(In the declarative method one implication per clause.)

Operational method proves more:
all the procedure calls satisfy the precondition
(often of no interest).

Operational method & our example

Specification:
\[
\begin{align*}
  \text{pre} &= \{ \text{app}(k,l,m) \mid l \text{ or } m \text{ is a list} \}, \\
  \text{post} &= \{ \text{app}(k,l,m) \mid k,l,m \text{ are lists, } k \cdot l = m \}. 
\end{align*}
\]

Comparison
operational and declarative methods
for proving correctness

Every operational specification-proof can be transformed into an equivalent \(^1\) declarative one.
\(^1\)as far as answers are concerned

The declarative method stronger
than the operational method.

Details:
Operational specification: \( \langle \text{pre}, \text{post} \rangle \).
Corresponding declarative specification:
\[
\text{pre} \rightarrow \text{post} \; := \; \{ A \in \mathcal{H} \mid A \in \text{pre} \rightarrow A \in \text{post} \}
\]

Theorem:
If \( P \) is correct w.r.t. operational specification
\( \langle \text{pre}, \text{post} \rangle \) then it is correct w.r.t. declarative specification \( \text{pre} \rightarrow \text{post} \).

If there exists a proof of the former by the operational method then there exists a proof of the latter by the declarative method.

Comparison, treatment of "ill-typed" atoms

<table>
<thead>
<tr>
<th>correctness</th>
<th>operational method</th>
<th>declarative method</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclude them (by precondition)</td>
<td>don't bother, allow them</td>
<td>Ex.: spec \models \text{app}(1,2,3)</td>
</tr>
<tr>
<td>( M_p \cap \text{pre} \subseteq \text{post} )</td>
<td>( M_p \subseteq \text{spec} )</td>
<td></td>
</tr>
</tbody>
</table>

dealing also with completeness

| \( M_p \cap \text{pre} = \text{post} \) | \( \text{spec2} \subseteq M_p \subseteq \text{spec} \) |
[Comparison, proving correctness]

The operational method bound to Prolog selection rule.

The method is inapplicable if procedure calls in LD-resolution do not satisfy the precondition; like
- other selection rule intended;
- "two pass" programs, difference lists, "logical variables", ...

Example (abstraction of a two pass compiler)

\[ p(X, Y) \leftarrow q(X, W, Z, Y), q(Z, W, -). \]
\[ q(X, Y, X, Y). \]

Declarative specification:

\[ \{ p(a, b) \in H \mid \text{list}(a) \rightarrow \text{list}(b) \} \]
\[ \bigcup \{ q(a, b, c, d) \in H \mid \text{list}(a) \rightarrow \text{list}(c), \text{list}(b) \rightarrow \text{list}(d) \} \]

[Example, contd.]

Corresponding precondition for \( q \)

\[ \{ q(a, b, c, d) \in H \mid \text{list}(a), \text{list}(b) \} \]

does not hold!
(as \( W \) is not a list at call of \( q(X, W, Z, Y) \)).

Note that using different \( (\text{pre}, \text{post}) \) for the two usages of \( q \) does not help.

Correctness proof, declarative method; outline:

\[ \text{list}(X) \Rightarrow \text{list}(Z) \Rightarrow \text{list}(W) \Rightarrow \text{list}(Y) \]

Conclusion: declarative method applicable, operational not

(Unless we make the preconditions true and use the declarative specification as the postconditions, i.e. unless we reduce the operational method to declarative)

[Comparison, proving correctness]

Declarative method strictly stronger in a sense

The declarative method can be seen as a special case of the operational method (with the preconditions true).

BUT

We show that for certain classes of specifications the operational method is weaker. Let us begin from some class of types (sets of terms). Consider sets of atoms of the form

\[ p(\text{type}_1, \ldots, \text{type}_n) := \{ p(u_1, \ldots, u_n) \mid u_i \in \text{type}_i \}. \tag{1} \]

Consider generalized operational specifications

\[ (\text{pre}_1, \text{post}_1), \ldots, (\text{pre}_k, \text{post}_k) \tag{2} \]

with \( \text{pre}_1, \ldots, \text{post}_k \) like in (1). Their meaning: each call is in some \( \text{pre}_i \) and the corresponding success is in \( \text{post}_i \). The operational method can be generalized for such specifications.

Their declarative counterpart is of the form

\[ (\text{pre}_1 \rightarrow \text{post}_1) \land \ldots \land (\text{pre}_k \rightarrow \text{post}_k) \tag{3} \]

In our last example, a declarative proof exists for the declarative specification (of the form (3)), but there does not exist a specification (2) for which an operational proof exists.
Conclusions

Clear separation between declarative & operational properties.
(correctness, completeness, vs. termination, call patterns, complexity, ...)

We can do a lot with declarative thinking.

Declarative proof methods are simple and intuitive.

Explicit notions of call patterns, modes, ill-typed queries,... are unnecessary (in reasoning about declarative properties).

Important: separate specifications for correctness and completeness. Notice 3-valued flavour.

My belief:
The advocated proof methods (possibly treated informally) are a valuable tool for programmers.
(And they are a formalization of a style of thinking of good logic programmers).

They should be used in teaching logic programming.

Comments

Declarative approach is applicable to techniques like accumulators, difference lists, etc (despite they are sometimes claimed to be not declarative).

The specification for correctness can be used instead of the program's model in termination proving method of Apt&Pedreschi.

Related approaches to completeness proving: [Deransart, Maluszynski '93], [Pedreschi, Ruggeri '99]. We prove completeness "modulo termination", this is simpler.

Some ideas similar to those presented here appeared in the work of Naish.

Other related work?