It Is Declarative
On Reasoning about Logic Programs

Włodzimierz Drabent
IPI PAN, Ordona 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) correctness by a method of proving completeness. We point out that usually one is interested in soundness and correctness w.r.t. two *different* specifications. 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

[Apt97] K. R. Apt. *From Logic Programming to Prolog.* Prentice Hall, 1997.

[Cla79] K. L. Clark. Predicate logic as computational formalism. Technical Report 79/59, Imperial College, London, December 1979.

[DM88] W. Drabent and J. Małuszyński. Inductive Assertion Method for Logic Programs. *Theoretical Computer Science*, 59:133–155, 1988.

[DM93] P. Deransart and J. Małuszyński. *A Grammatical View of Logic Programming.* The MIT Press, 1993.

[PR99] D. Pedreschi and S. Ruggieri. Verification of logic programs. *Journal of Logic Programming*, 39(1-3):125-176, April 1999.

It Is Declarative

On reasoning about logic programs

Włodzimierz Drabent

Polish Academy of Sciences
&
Linköping University

www.ipipan.waw.pl/~drabent

Unnecessarily complicated proof methods proposed in the literature.

We point out that simpler and – in some sense – more general methods exist.

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 $P \models Q$
(by soundness and completeness of SLD-resolution).

Total correctness

=

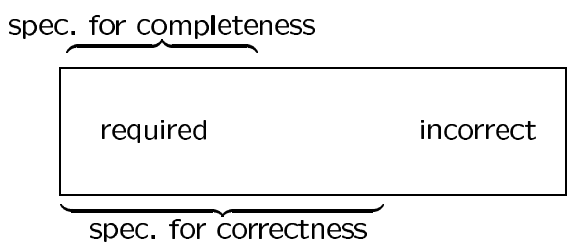
(partial) correctness (all the answers are correct) + completeness (all the required answers are answers)

+ termination

Correctness & completeness are independent from the operational semantics (in particular from call patterns).

Specifications

Different specifications
for correctness and completeness !



A specification may be: an interpretation,
a theory.

Example. Specification of append:

1. for correctness

$$spec = \left\{ app(k,l,m) \in \mathcal{H} \;\middle|\; \begin{array}{l} \text{if } l \text{ or } m \text{ is a list then} \\ k,l,m \text{ are lists} \\ \text{and } k * l = m \end{array} \right\}$$

($\mathcal{H}$ – Herbrand universe,
$*$ – list concatenation.)

2. for completeness

$$spec2 = \left\{ app(k,l,m) \in \mathcal{H} \;\middle|\; \begin{array}{l} k,l,m \text{ are lists} \\ \text{and } k * l = m \end{array} \right\}$$

Declarative Proof Method for Correctness

([Clark 79] or earlier)

P – Horn clause program
Q – computed instance of a query

$$SPEC \models Q \quad \text{if} \quad SPEC \models P$$

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

Specification: $spec$ from the previous example.

We have to show

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

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

Thus $[h|k] * l = [h|m]$. So $spec \models app([h|k], l, [h|m])$.

[proving correctness]

The method is *declarative*

program:	implications
computed instance:	formula Q
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 specification of a predicate is:

$$p(\vec{x}) \leftrightarrow \bigwedge_i \varphi_i \rightarrow \psi_i$$

To prove: implications of the form

$$\left(\bigwedge \varphi_j^k \rightarrow \psi_j^k \right) \rightarrow (\varphi_i \rightarrow \psi_i)$$

Proving completeness

Technicalities

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

For each predicate symbol p, if P contains

$$p(\vec{t}_1) \leftarrow B_1$$
$$\cdots$$
$$p(\vec{t}_k) \leftarrow B_k$$

then ONLY-IF(P) contains

$$p(\vec{x}) \rightarrow \bigvee_{i=1}^{k} \exists_{-\vec{x}}\, \vec{x} = \vec{t}_i \wedge B_i.$$

2.

$$speceq \;:=\; \{\, t = t \mid t \text{ is a ground term} \,\}.$$

Theorem: P a program, Q a query. If

(i) $spec2 \cup speceq \models$ ONLY-IF(P),

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

Then

1. if $spec2 \models \exists Q$ then $P \models \exists Q$ (and Q succeeds with P).

2. if $spec2 \models Q$ then $P \models Q$ (and Q is an answer of P).

Example. Program APPEND:

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

ONLY-IF(APPEND):

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

$$spec2 = \left\{ app(k,l,m) \in \mathcal{H} \;\middle|\; \begin{array}{l} k, l, m \text{ are lists} \\ \text{and } k * l = m \end{array} \right\}$$

We have $cspec2 \cup speceq \models$ ONLY-IF(APPEND)

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

$spec2 \models \exists Q$, thus Q succeeds.

Let k, l be lists such that $k{*}l = m$; $Q' = app(k, l, m)$.
Then $spec2 \models Q'$. 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,Małuszyński88]

It considers procedure calls & successes in LD-resolution.

Specifications: pre- & postconditions

Correctness: calls $\subseteq$ precondition,
successes $\subseteq$ 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:

$pre = \{\, app(k,l,m) \mid l$ or m is a list $\}$,
$post = \{\, app(k,l,m) \mid k,l,m$ are lists, $k * l = m \,\}$.

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: $\langle pre, post \rangle$.
Corresponding declarative specification:

$$pre \rightarrow post \;:=\; \{\, A \in \mathcal{H} \mid A \in pre \rightarrow A \in post \,\}$$

Theorem:
If P is correct w.r.t. operational specification $\langle pre, post \rangle$ then it is correct w.r.t. declarative specification $pre \rightarrow post$.
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

correctness

operational method	declarative method
exclude them (by precondition)	don't bother, allow them Ex.: $spec \models app(1,2,3)$
$\mathcal{M}_P \cap pre \subseteq post$	$\mathcal{M}_P \subseteq spec$

dealing also with completeness

$\mathcal{M}_P \cap pre = post$	$spec2 \subseteq \mathcal{M}_P \subseteq spec$

[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:

$$\{\, p(a,b) \in \mathcal{H} \mid list(a) \rightarrow list(b) \,\}$$
$$\cup$$
$$\left\{ q(a,b,c,d) \in \mathcal{H} \;\middle|\; \begin{array}{l} list(a) \rightarrow list(c) \\ list(b) \rightarrow list(d) \end{array} \right\}$$

[Example, contd.]

Corresponding precondition for q

$$\{\, q(a,b,c,d) \in \mathcal{H} \mid list(a), list(b) \,\}$$

does not hold!
(as W is not a list at call of $q(X,W,Z,Y)$).
Note that using different $\langle pre, post \rangle$ for the two usages of q does not help.

Correctness proof, declarative method; outline:

$$list(X) \Rightarrow list(Z) \Rightarrow list(W) \Rightarrow list(Y)$$

Conclusion: **declarative method applicable, operational not**

(Unless we make the preconditions **true** and use the declarative specification as the post-conditions, 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

$$p(type_1, \ldots, type_n) := \{\, p(u_1, \ldots, u_n) \mid u_i \in type_i \,\}. \quad (1)$$

Consider generalized operational specifications

$$\langle pre_1, post_1 \rangle, \ldots, \langle pre_k, post_k \rangle \quad (2)$$

with $pre_1, \ldots, post_k$ like in (1). Their meaning: each call is in some pre_i and the corresponding success is in $post_i$. The operational method can be generalized for such specifications.

Their declarative counterpart is of the form

$$(pre_1 \rightarrow post_1) \wedge \ldots \wedge (pre_k \rightarrow post_k) \quad (3)$$

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, Ruggieri '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 ?