It is declarative
On declarative programming in Prolog

Włodzimierz Drabent

Institute of Computer Science, Polish Academy of Sciences (IPI PAN);
IDA, Linköpings universitet, Sweden
www.ipipan.waw.pl/~drabent/

LOPSTR 2021

Version 1.0 compiled September 14, 2021

This file includes examples, slides and slide overlays not used in the presentation

Logic Programming (LP)

introduced as a declarative programming paradigm

Prolog – implementation of LP

However it seems
the declarative aspect is often neglected, or diminished

Ex.: Now, we compute the factorial usign bottom up method so we start with the trivial problem of computing the factorial of 0 and continue with the factorial of 1, 2 and so on till the factorial of N is known. [...] we [...] store the computed facts using additional parameters. [...] we remember [...] the factorial of M in the M-th step.

```
fact_bu(N,F):-fact_bu1(0,1,N,F).
fact_bu1(N,F,N,F).
fact_bu1(N1,F1,N,F):-
  N1<N, N2 is N1+1, F2 is N2*F1, fact_bu1(N2,F2,N,F).
```

(From a Prolog tutorial [Barták'98])

Declarative descriptions:

% fact_bu1(N', F', N, F) – if $0 \leq N' \leq N$ and $F' = N'!$ then $F = N!$

or – $F = F' * (N'{+}1) * (N'{+}2) * \cdots * N$

We do not understand a program
without understanding the relations it defines.

This talk

A look at the basics of LP

LP in Prolog

Practical Prolog programming can be declarative

or

Prolog can be used for LP

to an extent larger than usually supposed/understood/meant.

Program correctness in LP

Imperative programming :	partial correctness
	↙ ↘
LP :	correctness completeness

correctness = the answers of the program are as required
completeness = all required answers are answers of the program

Df.: correctness$^+$ = correctness + completeness
(full correctness?)
(double correctness?)

Outline — main issues of the talk

Introduction; basic notions.

1. Reasoning (declaratively) about correctness$^+$ of programs.
   Role of approximate specifications.
2. A systematic way of constructing correct$^+$ programs.
   from specifications.
   Limitations of semantics preserving program transformations.
3. Declarative diagnosis (aka. algorithmic debugging) made useful.

Introduction

- Basics of LP
- Specifications
- Correctness and completeness
- Examples
- Approximate specifications

Basics of LP

terminology clash
logic ↔ Prolog

Df.: Query – conjunction $A_1, \ldots, A_n$ of atoms (atomic formulae)

Program – set of clauses $A_0 \leftarrow A_1, \ldots, A_n$,

Answer of a program P – query Q such that $P \models Q$
(correct answer)

computed answer substitution

SLD-resolution – obtaining answers $Q\theta$ from an initial query Q

Computed vs. correct answers?
We do not need to distinguish them
$\leadsto$ soundness and completeness of SLD-resolution

Notation

P a theory	$P \models A$ – A logical consequence of P
S an interpretation	$S \models A$ – A true in S

$\mathcal{HB}$ (Herbrand base) – the set of ground atoms

$\mathcal{M}_P = \{ A \in \mathcal{HB} \mid P \models A \}$ – the least Herbrand model of P

Specifications

LP – relational programming.

A logic programmer has to understand the relations defined by her program.

Specification – should describe for each predicate symbol
a relation on ground terms. So:

Df.: Specification – Herbrand interpretation $S \subseteq \mathcal{HB}$
(i.e. a set of ground atoms).

The relation for p: $[\![p]\!] = \{ \vec{t} \mid p(\vec{t}) \in S \}$

Ex.:
$S^0_{\rm member} = \{ mem(e_i, [e_1, \ldots, e_n]) \in \mathcal{HB} \mid 1 \leq i \leq n \}$

List membership

Note

crucial for program understanding

Specifications (in LP)
play the role of loop invariants (in imperative programming)
or assertions

"understanding a loop means understanding its invariant"
(maybe without explicitly referring to this notion)
[Furia,Meyer,Velder'14 ACM C.Surveys]
[Dijkstra'??]

A bit of code:

$\ldots$
$\ldots A[i] \ldots$ $\longleftarrow$ Is i here the number of the last already processed element of A? Or the first unprocessed one?
$\ldots$

On programmers who have not learnt about invariants:
if they understand what they are doing they are relying on some intuitive understanding of the invariant anyway, like Molière's Mr. Jourdain speaking in prose without knowing it.

Note

crucial for program understanding

Specifications (in LP)

play the role of loop invariants (in imperative programming)

or assertions

"understanding a loop means understanding its invariant"

(maybe without explicitly referring to this notion)

[Furia,Meyer,Velder'14 ACM C.Surveys]

[Dijkstra'??]

LP: understanding a program = understanding the relations it defines

Correctness$^+$ of programs

Let S – a specification, P – a program.

Df.: P is correct w.r.t. S when $\mathcal{M}_P \subseteq S$.
P is complete w.r.t. S when $S \subseteq \mathcal{M}_P$.

Declarative notions,
independent from any operational semantics

LOGIC + CONTROL works, as
correctness$^+$ independent from CONTROL

Details, answers of correct / complete programs

Non-atomic, non-ground answers

Th.: P correct w.r.t. Q: Q an answer of $P \;\Rightarrow\; S \models Q$.
P complete w.r.t. Q: $S \models Q \;\Rightarrow\; Q$ an answer of P,
when Q ground, or the alphabet of function symbols infinite, or...

Ex.: (the extra conditions at completeness)

Alphabet $\{f/1, a/0\}$, $P = \{p(f(X)).\;\; p(a).\}$, $S = \mathcal{HB} = \mathcal{M}_P$.
P complete w.r.t. S.
$S \models p(Y)$, but $p(Y)$ is not an answer of P.

Examples (specifications, correctness, completeness)

Appending lists

$$S^0_{\rm app} = \{\, app(k,l,m) \in \mathcal{HB} \;\mid k,l,m \text{ are lists, } k\hat{\ }l = m \,\},$$

^ means list concatenation.

(The same in another notation:
$\{\, app([x_1,\dots,x_k],[y_1,\dots,y_m],[x_1,\dots,x_k,y_1,\dots,y_m]) \in \mathcal{HB} \mid k,m \in \mathbb{N} \,\}$)

Standard program APP: $app(\,[\,], L, L\,).$
$app(\,[H|K], L, [H|M]\,) \leftarrow app(\,K, L, M\,).$

APP complete w.r.t. $S^0_{\rm app}$, but not correct. APP $\models app([\,],6,6)$

APP does not define the list appending relation ($\mathcal{M}_{\rm APP} \neq S^0_{\rm app}$).
(There are even opinions that APP is a wrong program.
It is not, see the next slide.)

Example (cont'd)

APP correct w.r.t. the following specifications

$$S_{\text{app},1} = \left\{ app(k,l,m) \in \mathcal{HB} \;\middle|\; \begin{array}{l}\text{if } k \text{ and } l \text{ are lists}\\ \text{then } m \text{ is a list}\\ \text{and } k\hat{~}l = m\end{array} \right\}$$

for list appending

$$S_{\text{app},2} = \left\{ app(k,l,m) \in \mathcal{HB} \;\middle|\; \begin{array}{l}\text{if } m \text{ is a list}\\ \text{then } k \text{ and } l \text{ are lists}\\ \text{and } k\hat{~}l = m\end{array} \right\}$$

for list splitting

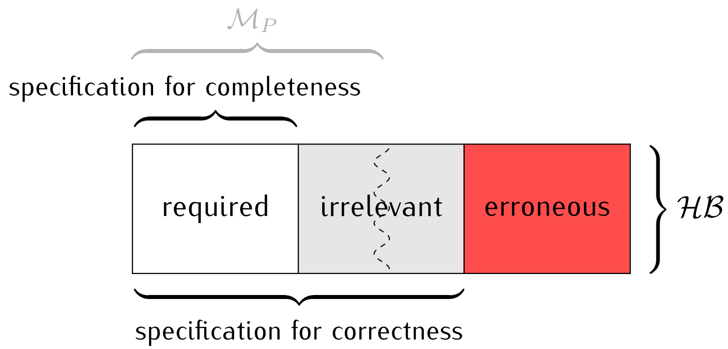
$$S_{\text{app}} = \left\{ app(k,l,m) \in \mathcal{HB} \;\middle|\; \begin{array}{l} k \text{ is a list,}\\ \text{if } l \text{ or } m \text{ is a list}\\ \text{then } l, m \text{ are lists}\\ \text{ and } k\hat{~}l = m\end{array} \right\}$$

more precise, for most usages

$S_{\text{app}} \subset S_{\text{app},1} \cap S_{\text{app},2}$

Approximate specifications

Usual situation:



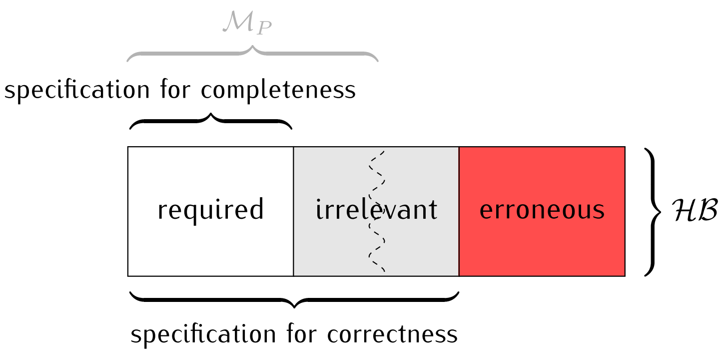
Approximate specification: (S_{compl}, S_{corr})

Correctness+: $S_{compl} \subseteq \mathcal{M}_P \subseteq S_{corr}$

$\mathcal{M}_P$ may differ in different programs for the same task or at various stages of program development

Approximate specifications

Usual situation:



Approximate specification: (S_{compl}, S_{corr})

Correctness+: $S_{compl} \subseteq \mathcal{M}_P \subseteq S_{corr}$

When we build a program,
not known in advance if a given $A \in S_{compl} \setminus S_{corr}$ is in $\mathcal{M}_P$

Approximate specification, example, insertion sort

Ex. (we cannot know in advance, if $A \in \mathcal{M}_P$):

$insert/3$ – inserting a number into a sorted list

Should we accept $A = insert(2, [3, 1], [2, 3, 1])$? It's **irrelevant**!

Approximate specification: $(S^0_{insert}, S_{insert})$, $A \in S_{insert} \setminus S^0_{insert}$

$$S_{insert} = \left\{ \begin{array}{r} insert(n, l_1, l_2)\\ \in \mathcal{HB}\end{array} \;\middle|\; \begin{array}{l} n \notin \mathbb{Z}, \text{ or}\\ l_1 \text{ not a sorted list}\\ \text{of integers}\end{array} \right\} \cup S^0_{insert}$$

Approximate specification, example, insertion sort

Ex. (we cannot know in advance, if $A \in \mathcal{M}_P$):

$insert/3$ – inserting a number into a sorted list

Should we accept $A = insert(2, [3,1], [2,3,1])$? It's **irrelevant**!

Approximate specification: $(S^0_{insert}, S_{insert})$, $A \in S_{insert} \setminus S^0_{insert}$

$$S^0_{insert} = \left\{ \begin{array}{r|l} insert(n, l_1, l_2) & l_1, l_2 \text{ are sorted lists of integers,} \\ \in \mathcal{HB} & elms(l_2) = \{n\} \cup elms(l_1) \end{array} \right\}$$

where $elms(l)$ – the multiset of elements of l

$$S_{insert} = \left\{ \begin{array}{r|l} & \text{if } n \in \mathbb{Z} \text{ and} \\ insert(n, l_1, l_2) & l_1 \text{ is a sorted list of integers,} \\ \in \mathcal{HB} & \text{then } insert(n, l_1, l_2) \in S^0_{insert} \end{array} \right\}$$

Reasoning (declaratively) about correctness$^+$ of programs

- Proving correctness
- Proving completeness

Proving program correctness

Th. [Clark'79]: (the simplest theorem of LP ☺)
Let S – a specification, P – a program.

$$\boxed{\text{If } S \models P \ \text{ then } \ P \text{ correct w.r.t. } S.}$$

Proof: $S \models P \Rightarrow \mathcal{M}_P \subseteq S$ □

Note: $S \models P$ means
for each ground instance $H \leftarrow B_1, \ldots, B_n$ of a clause of P
if $B_1, \ldots, B_n \in S$ then $H \in S$

The Th. – a declarative way to prove a declarative property.

The Th. should be well-known, but is unacknowledged.

Instead, more complicated methods based on operational semantics, on pre- and postconditions for LD-resolution [Bossi+Cocco'89,Apt'97,...].

Example correctness proof

$$\boxed{\text{For each } H \leftarrow B_1, \ldots, B_n \in ground(P), \text{ if } B_1, \ldots, B_n \in S \text{ then } H \in S.}$$

Program + specification:

SPLIT:

$$s([\,], [\,], [\,]). \quad (1)$$
$$s([X|Xs], [X|Ys], Zs) \leftarrow s(Xs, Zs, Ys). \quad (2)$$

$$S = \{\, s(l, l_1, l_2) \mid l, l_1, l_2 \text{ are lists, } 0 \leq |l_1| - |l_2| \leq 1 \,\},$$

where $|l|$ – the length of a list l.

Proof:
Consider a ground instance $s([h|t], [h|t_2], t_1) \leftarrow s(t, t_1, t_2)$ of (2).
Assume $s(t, t_1, t_2) \in S$. Thus $[h|t], [h|t_2], t_1$ are lists. Let $m = |t_1| - |t_2|$.
As $m \in \{0,1\}$, we have $|[h|t_2]| - |t_1| = 1 - m \in \{0,1\}$.
So the head $s([h|t], [h|t_2], t_1)$ is in S. The proof for (1) is trivial.

Thus program SPLIT correct w.r.t. specification S.

Example correctness proof 2

$\boxed{\text{If } S \models P \text{ then } P \text{ correct w.r.t. } S.}$

We need to show:
for each $H{\leftarrow}B_1,\ldots,B_n \in ground(P)$ if $B_1,\ldots,B_n \in S$ then $H \in S$

$$S'_{\rm app} = \left\{ app(k,l,m) \in \mathcal{HB} \;\middle|\; \begin{array}{l} \text{if } l \text{ or } m \text{ is a list then} \\ k,l,m \text{ are lists and } k\hat{\ }l = m \end{array} \right\}$$

APP: $app(\,[\,], L, L\,).\qquad app(\,[H|K], L, [H|M]\,) \leftarrow app(\,K, L, M\,).$

Nontrivial part of a correctness proof for APP w.r.t. $S'_{\rm app}$:

Take a ground $\overbrace{app([h|k], l, [h|m])}^{H} \leftarrow \overbrace{app(k,l,m)}^{B}$, assume $B \in S'_{\rm app}$;
assume l or $[h|m]$ is a list, show that $[h|k]\hat{\ }l = [h|m]$; so $H \in S_{\rm app}$.
(l or m is a list $\Rightarrow$ k,l,m are lists $\Rightarrow$ $k\hat{\ }l = m$)

Similar to informal reasoning about a program
by a competent declarative programmer.

Example correctness proof 3

MIDDLE:
$$\begin{array}{ll} middle(Mid, L) \leftarrow m(Mid, L, L). & (1) \\ m(E, [\_], [E|\_]). & (2) \\ m(E, [\_,\_|L1], [\_|L2]) \leftarrow m(E, L1, L2). & (3) \end{array}$$

$$\begin{aligned} S_{\rm M} = \;& \{\, middle(b_i, [b_1,\ldots,b_{2i-1}]) \in \mathcal{HB} \mid i > 0 \,\} \\ \cup\; & \{\, m(t,l,t') \in \mathcal{HB} \mid t' \text{ is not a list} \,\} \\ \cup\; & \{\, m(b_i, [a_1,\ldots,a_{2i-1}], [b_1,\ldots,b_n]) \in \mathcal{HB} \mid n \geq i > 0 \,\} \end{aligned}$$

$\boxed{\text{If } S \models P \text{ then } P \text{ correct w.r.t. } S.}$

We need to show:
for each $H{\leftarrow}B_1,\ldots,B_n \in ground(P)$ if $B_1,\ldots,B_n \in S$ then $H \in S$.

Non-obvious part of a correctness proof for MIDDLE w.r.t. $S_{\rm M}$

Take a ground instance $H \leftarrow B$ of (3). Show that
if $B \in S_{\rm M}$ then $H \in S_{\rm M}$.

Example correctness proof 3 (cont'd)

$$m(E, [\_,\_|L1], [\_|L2]) \leftarrow m(E, L1, L2). \qquad (3)$$

$$\begin{aligned} S_{\rm M} = \;& \{\, middle(b_i, [b_1,\ldots,b_{2i-1}]) \in \mathcal{HB} \mid i > 0 \,\} \\ \cup\; & \{\, m(t,l,t') \in \mathcal{HB} \mid t' \text{ is not a list} \,\} \\ \cup\; & \{\, m(b_i, [a_1,\ldots,a_{2i-1}], [b_1,\ldots,b_n]) \in \mathcal{HB} \mid n \geq i > 0 \,\} \end{aligned}$$

Take a ground instance $\overbrace{m(e, [e_1, e_2|l_1], [e_3|l_2])}^{H} \leftarrow \overbrace{m(e, l_1, l_2)}^{B}$ of (3).
Assume $B \in S_{\rm M}$. Then
1. l_2 is not a list, thus $H \in S_{\rm M}$, or
2. $e = b_i$, $l_1 = [a_1,\ldots,a_{2i-1}]$, $l_2 = [b_1,\ldots,b_n]$, $n \geq i > 0$. Hence
$H = m(b_i, [e_1, e_2, a_1,\ldots,a_{2i-1}], [e_3, b_1,\ldots,b_n])$. Renumber it:
$H = m(b'_{i+1}, [a'_1,\ldots,a'_{2(i+1)-1}], [b'_1,\ldots,b'_{n+1}])$,
where $(n+1) \geq (i+1) > 0$.

Thus $H \in S_{\rm M}$. $\square$

Similar to informal reasoning about a program
by a competent declarative programmer.

Reasoning about program completeness

Surprising: the subject has been neglected ! $\ddot{\frown}$

Except for
[Deransart+Małuszyński'93], [Sterling+Shapiro'94] (informally),
[D_+Miłkowska'05], [D_'16,'18]; I am not aware of any other work.

[Hogger'84], [Kowalski'85] – the notion of completeness,
but not reasoning about it.

Semi-completeness

$$\boxed{\text{completeness} \;=\; \text{semi-completeness} \;+\; \text{termination}}$$

Df.: P is complete for a query Q w.r.t. S if for any ground $Q\theta$

$$S \models Q\theta \quad \Rightarrow \quad Q\theta \text{ is an answer for } P.$$

(P produces all the required answers for Q.)

Df.: P is semi-complete w.r.t. S if P is complete w.r.t. S for any query for which there exists a finite SLD-tree.

(P produces all the required answers, if the computation terminates.)

Lemma: If P is semi-complete w.r.t. S, and
P terminates (under some selection rule) for each query $A \in S$
then P is complete w.r.t. S.

Sufficient condition for completeness

$$\boxed{\text{completeness} \;=\; \text{semi-completeness} \;+\; \text{termination}}$$

Df.: $H \in \mathcal{HB}$ is covered by P w.r.t. S if there is a clause $(H \leftarrow A_1, \ldots, A_n) \in ground(P)$ in which $A_1, \ldots, A_n \in S$.

(A covered atom can be produced by a clause of P from atoms required by S to be produced.)

Th. (sufficient condition):
If each atom from S is covered w.r.t. S by P
then P is semi-complete w.r.t. S.

Proving program termination – not discussed here.

Example

$S^0_{\rm app} = \{\, app(k,l,m) \in \mathcal{HB} \mid k,l,m$ are lists, $k\hat{\ }l = m \,\}$,

APP: $app(\, [\,], L, L\,).$
$app(\, [H|K], L, [H|M]\,) \leftarrow app(\, K, L, M\,).$

Let $H \in S^0_{\rm app}$. We show that H is covered by APP w.r.t. $S^0_{\rm app}$.

1. $H = app([\,], l, l)$. $H \in ground(\text{APP})$
2. $H = app(k, l, m)$, $k \neq [\,]$, $k\hat{\ }l = m$. So H is the head of $app([h|k'], l, [h|m']) \leftarrow app(k', l, m')$ and $app(k', l, m') \in S^0_{\rm app}$.

Thus APP semi-complete w.r.t. $S^0_{\rm app}$.

We know that P terminates for any query from S.

Hence APP complete w.r.t. $S^0_{\rm app}$.

Reasoning about completeness, comments

Not fully declarative, as termination is an operational property.
(Proving semi-completeness purely declarative)

But termination has to be established anyway.

So the not fully declarative approach seems reasonable.

A declarative sufficient condition exists [Deransart+Małuszyński'93].
But it leads to completeness proofs similar to
proving semi-completeness + termination.

Semi-completeness alone: Computation terminates $\Rightarrow$
all required (by the specification) answers have been produced.

Proving correctness$^+$, comments

In my opinion

the sufficient conditions for correctness & (semi-) completeness

- are declarative – abstract from operational semantics
  (except for termination, which is needed anyway)
- are simple (cf. Hoare rules for imperative programming)
- correspond to a natural way of thinking by a declarative programmer
- can be used in every-day programming
  at various levels of (in)formality
- provide a guide how to reason about programs

Constructing correct$^+$ programs

How to construct a program
for an approximate specification $\mathcal{S} = (S_{compl}, S_{corr})$

Provide clauses so that

1. each atom $A \in S_{compl}$ is covered (w.r.t. S_{compl}) by some clause
2. each clause satisfies the sufficient condition for correctness (w.r.t. S_{corr})

(this produces a program correct and semi-complete w.r.t. $\mathcal{S}$;
not enough, $p(\vec{X}) \leftarrow p(\vec{X})$ possible)

3. the clauses satisfy some sufficient condition for termination
   informally: $p(\vec{s}) \leftarrow \ldots, p(\vec{t}), \ldots$
   $\uparrow$ bigger terms $\uparrow$ smaller terms

Result: a program correct and complete [D_'18]

A more interesting example – file `ex.insert*.pdf`

Constructing correct$^+$ programs, example

Splitting a list into its odd- and even- numbered elements.

$$S = \{\, s(l, oe(l), ee(l)) \in \mathcal{HB} \mid l \text{ is a list} \,\}$$

where $oe(l)$ – the list of odd elements of list l
$(\, oe([e_1, \ldots, e_n]) = [e_1, e_3, ...] \,)$
$ee(l)$ – the list of even elements of list l
(e.g. $s([1,2,3,4,5],[1,3,5],[2,4]) \in S$)

An unusual case of exact specification!

Construct a program correct$^+$ w.r.t. (S, S).

Constructing correct$^+$ programs, example

Summary of the approach: 1. each atom $A \in S_{compl}$ is covered
2. each clause correct w.r.t. S_{corr}
3. termination...

$S = \{ s(l, oe(l), ee(l)) \mid l \text{ is a list} \}$. Two kinds of elements of S:

1. $s([\,],[\,],[\,])$. Covered by clause $C_1 = s([\,],[\,],[\,])$. $S \models C_1$.
2. $A = s([h|t],[h|ee(t)],oe(t))$

We need a $B \in S$ for clause body.
Preferably subterms of arguments of A should be used (for termination).
What about $[h|t] \rightsquigarrow t$? $B = s(t, oe(t), ee(t))$?

This suggests $A \leftarrow B \in ground(P)$

Constructing correct$^+$ programs, comments

Summary of the approach: 1. each atom $A \in S_{compl}$ is covered
2. each clause correct w.r.t. S_{corr}
3. termination...

This proposal is rather obvious

I see it as
good practices of competent programmers made explicit

☞ Approximate specifications crucial

Beginning with exact specification $S_{compl} = S_{corr}$ is often unnecessary & counterproductive

We should not/cannot decide in advance
what $insert/3$ (of insertion sort) should do with unsorted lists
what $append/3$ should do with non-lists

Constructing correct$^+$ programs, example

Summary of the approach: 1. each atom $A \in S_{compl}$ is covered
2. each clause correct w.r.t. S_{corr}
3. termination...

$S = \{ s(l, oe(l), ee(l)) \mid l \text{ is a list} \}$. Two kinds of elements of S:

1. $s([\,],[\,],[\,])$. Covered by clause $C_1 = s([\,],[\,],[\,])$. $S \models C_1$.
2. $A = s([h|t],[h|ee(t)],oe(t))$

We need a $B \in S$ for clause body.
Preferably subterms of arguments of A should be used (for termination).
What about $[h|t] \rightsquigarrow t$? $B = s(t, oe(t), ee(t))$?

This suggests $C_2 = s([H|T],[H|ET],OT) \leftarrow s(T,OT,ET)$.
(Each) A covered by C_2. $S \models C_2$.

P terminates for any query $s([e_1,\dots,e_n],t,t')$ (maybe nonground).
$P = \{C_1, C_2\}$ correct & complete w.r.t. S.

On semantics-preserving program transformations

Program development: $P_1,\dots,P_n$ $\quad \forall i \; S_{compl} \subseteq \mathcal{M}_{P_i} \subseteq S_{corr}$

The programs may be not equivalent
– distinct relations for the same predicate in P_i, P_j
$$\{ q(\vec{t}) \in \mathcal{M}_{P_i} \} \neq \{ q(\vec{t}) \in \mathcal{M}_{P_j} \}$$

The paradigm of
semantics-preserving program transformations

too restrictive

Ex.: Construction of SAT-solver [D_'18, Howe+King'12]
P_1, P_2, P_3, P; distinct semantics of the main predicates in P_1, P_2.
↑
Prolog program

[D_'18] illustrates the methods presented here + ...

More precisely

Program development: $P_1, \ldots, P_n \quad \forall i \; S_{compl,i} \subseteq \mathcal{M}_{Pi} \subseteq S_{corr,i}$

The specification is constant for some main predicates, so

$$\forall i \; S_{compl,q} \subseteq \{ \, q(\vec{t}) \in \mathcal{M}_{Pi} \, \} \subseteq S_{corr,q}$$

Declarative diagnosis (algorithmic debugging)

All the declarativeness gone, when it comes to debugging

Next file of slides