It's declarative
On declarative programming in Prolog
2nd part

(includes many extra slides)

Włodzimierz Drabent

Version 1.0, compiled September 14, 2021

Declarative Diagnosis (DD)

locating errors in programs, declaratively

An observation

Debugging at the periphery of teaching or research

Often
one teaches a programming language
without teaching programming;
even when one teaches programming
one does not teach debugging.
[M.Ducassé]

Debugging – difficult to teach, to study, to find example buggy programs.

Here we discuss diagnosis, i.e. locating errors in programs.

(Debugging = diagnosis + error correction)

Declarative diagnosis (algorithmic debugging)

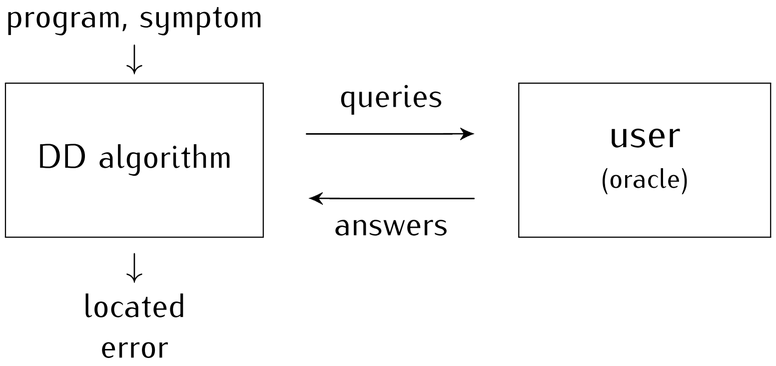
All the declarativeness gone, when it comes to debugging ☹

The Prolog debugger – purely operational;
worse, declarative-programmer-unfriendly:
information needed by a declarative-programmer
difficult to obtain from the debugger [D_'20 LOPSTR]

Declarative Diagnosis (DD), a.k.a. algorithmic debugging
[Shapiro'83,Pereira'86,Naish...,Nadjm-Tehrani et al'89,...]
Abandoned; no available tools.

We explain **why** DD has been abandoned
how to use DD effectively [D_'16]

DD (Declarative Diagnosis)



Queries – the intended declarative semantics of the program

User can locate the error without looking at the program
solely in terms of declarative semantics

DD, roughly

A symptom for incorrectness – a wrong answer
for incompleteness – a missing answer

An error – the/a reason that
the sufficient condition for incorrectness / incompleteness does not hold

In the program, a clause / a procedure corresponds to the incorrectness / incompleteness error

Diagnosis – search of a proof tree / SLD-tree for an incorrectness / incompleteness error

BTW diagnosis by proof failure possible (without symptoms)
– a failed proof attempt can show why the sufficient condition is violated

Incorrectness diagnosis

P – program, S – specification. P not correct w.r.t. S.

Symptom (incorrect answer) – atom A such that

$$P \models A \quad \text{but} \quad S \not\models A$$

Error (the/a reason of incorrectness) – an incorrect clause:
a $C \in P$ such that $S \not\models C$,

Notice: no errors $\Rightarrow$ the program correct

Incorrectness diagnosis algorithm: Given a symptom, finds an error.

Asks questions about atoms, $S \overset{?}{\models} B$.

Main idea – search of the proof tree for symptom A.

Incorrectness diagnosis algorithm

Given a symptom, find an error.
Search of the proof tree for symptom A.

Algorithm: Start at the root A.

- $S \models B_i$ for each child[1] B_i of A $\Rightarrow$ error found,
- $S \not\models B_j$ $\Rightarrow$ search the subtree with root B_j.

---

[1] This includes the case of no children

Incorrectness diagnosis, example [Shapiro'83]

- A specification (for correctness) for insertion sort:

$$S = \left\{ isort(l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} l' \text{ is a sorted permutation} \\ \text{of a list } l \text{ of numbers} \end{array} \right\} \cup$$

$$\left\{ insert(n,l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} \text{if } n \text{ is a number and} \\ \quad l \text{ is an ordered list of numbers} \\ \text{then } l' \text{ is } l \text{ with } n \text{ added and is ordered} \end{array} \right\}$$

$$\cup \{ i > j \mid \ldots \} \cup \ldots$$

Incorrectness diagnosis, example cont'd

The algorithm asked questions about some atoms in the proof tree, and found the error (incorrect clause instance):

```
insert(1,[3],[3,1]) :- 3 > 1, insert(1,[],[1]).
```

The clause in the program:

```
insert(X,[Y|Ys],[Y|Zs]) :- Y > X, insert(X,Ys,Zs).
```


Incorrectness diagnosis, example [Shapiro'83]

- A specification (for correctness) for insertion sort:

$$S = \left\{ isort(l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} l' \text{ is a sorted permutation} \\ \text{of a list } l \text{ of numbers} \end{array} \right\} \cup$$

$$\left\{ insert(n,l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} \text{if } n \text{ is a number and} \\ \quad l \text{ is an ordered list of numbers} \\ \text{then } l' \text{ is } l \text{ with } n \text{ added and is ordered} \end{array} \right\} \cup \ldots$$

- The program answers $Y = [2,3,1]$ for $isort([2,1,3], Y)$.
- Proof tree:

$$isort([2,1,3],[2,3,1])$$

$$isort([1,3],[3,1]) \qquad insert(2,[3,1],[2,3,1])$$

$$isort([3],[3]) \qquad insert(1,[3],[3,1]) \qquad \cdots$$

$$\cdots \qquad 3 > 1 \qquad insert(1,[],[1])$$

Incorrect (w.r.t. S) atoms marked red, incorrect clause instance red and blue
Error found without looking at the program!

Incorrectness. On the notion of error

An error – incorrect clause.

(The algorithm gives incorrect clause *instance*,
in a sense, more informative than a clause.)

More precise error location – impossible.
We cannot state which atom of the clause is wrong.

Ex.:

```
insert(X,[Y|Ys],[Y|Zs]) :- Y > X, insert(X,Ys,Zs).
```

may be corrected as

```
insert(X,[Y|Ys],[Y|Zs]) :- Y < X, insert(X,Ys,Zs).
```

or

```
insert(Y,[X|Ys],[X|Zs]) :- Y > X, insert(Y,Ys,Zs).
```

Incompleteness diagnosis

Program P not complete w.r.t. S^0 i.e. $S^0 \not\subseteq \mathcal{M}_P$

Incompleteness symptom: An atomic query A
for which some answer required by S^0 has not been produced
despite a finite SLD-tree.

Incompleteness error: A not covered atom $B \in S^0$
(reason of incompleteness) by P w.r.t. S^0

An error $p(\ldots)$ locates whole procedure (predicate definition) p.
More precise locating – impossible.

Diagnosis algorithm, roughly
extracts from SLD-tree atomic queries with their answers.
Search for one which is a symptom and does not depend on other symptoms.
Questions: Is $A, A\theta_1, \ldots, A\theta_n$ a symptom?

Incompleteness diagnosis, example

- A specification (for completeness) for insertion sort:

$$S^0 = \left\{ isort(l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} l' \text{ is a sorted permutation} \\ \text{of a list } l \text{ of numbers} \end{array} \right\} \cup$$

$$\left\{ insert(n,l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} l \text{ is an ordered list of numbers, } n \text{ is a number} \\ l' \text{ is } l \text{ with } n \text{ added and is ordered} \end{array} \right\} \ldots$$

- Query $A = isort([4,1,3], L)$ fails with the same `isort` program
- Incompleteness questions asked and answered about:
  (Y – yes, some answers are missing; N – no)

$isort([1,3], Zs)$	with answers $Zs = [3,1]$, $Zs = [1,3]$	N
$insert(4, [3,1], L)$	no answers	N
$A_3 = insert(4, [1,3], L)$	no answers	Y
$1 > 4$ no answers	$4 =< 1$ no answers	N N

- A_3 found, some its instance $A_3\theta$ is an error
  $A_3\theta$ uncovered by P w.r.t. S^0,

Incompleteness diagnosis, example

- A specification (for completeness) for insertion sort:

$$S^0 = \left\{ isort(l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} l' \text{ is a sorted permutation} \\ \text{of a list } l \text{ of numbers} \end{array} \right\} \cup$$

$$\left\{ insert(n,l,l') \in \mathcal{HB} \;\middle|\; \begin{array}{l} l \text{ is an ordered list of numbers, } n \text{ is a number} \\ l' \text{ is } l \text{ with } n \text{ added and is ordered} \end{array} \right\}$$

$$\cup \{ i > j \mid \ldots \} \cup \ldots$$

Comments

- Incorrectness: Error – a clause.
  Incompleteness: Error – a procedure (predicate definition).
  More precise diagnosis – impossible.
- Often: incorrectness and incompleteness occur together.
  Wrong answers instead of correct ones.
  When incorrectness found during incompleteness diagnosis
  (like $isort([1,3], Zs) \rightsquigarrow Zs = [3,1]$, $Zs = [1,3]$)
  – switch to incorrectness diagnosis.
- Crucial: a possibility of using approximate specifications.

Reasons for DD being neglected

- No freedom: fixed order of queries to answer
- . . .
- Exact specification (*intended model*) required from the user
  But she does not know it (and it does not matter)
  E.g. `member`(e, t) for a non-list t,
  `append`(l, t, t') for non-lists t, t',
  `insert`(e, l, y) in insertion sort, for unsorted l,

The user knows an approximate specification (S_{compl}, S_{corr})

The standard Declarative Diagnosis works!

when instead of the intended model we use

- S_{corr} for incorrectness diagnosis
- S_{compl} for incompleteness diagnosis

No need for *inadmissible* atoms, *3-valued DD*,. . .
[Pereira'86, Naish'00,. . .]

The standard Declarative Diagnosis works
with approximate specifications!

Seems an obvious observation, but somehow unnoticed

The state of Prolog debugging, lack of DD tools – harmful
Debugging must be operational ⇒ the advantages of LP disappear

DD tools easy to construct.
Future work: incompleteness diagnosis for other selection rules (delays)

We have simple&naive prototypes, useful in many cases
including debugging themselves

Experience: DD can substantially simplify locating errors
A proof tree browser – a useful incorrectness diagnoser

Dear Prolog vendors, DD tools, please!

Future work

Formalization of specifications,
automating proof checking / proving

Programs with negation (but see [D_&Miłkowska'05])

Implementing DD tools

Experimenting, teaching

Summary

We focus on declarative programming;
prefer abstracting from any operational semantics.

- Reasoning on correctness$^+$ independently of any operational semantics.
  (with a minor exception)
  Simple methods. Can be used (informally) in practical programming.
- Importance of approximate specifications.
  Intended model considered harmful.
  We did not need types.
- Approximate specifications make declarative diagnosis useful.
- A simple approach of constructing provably correct$^+$ programs.
  Can be used (informally) in practical programming.
- Semantics-preserving program transformations – too restrictive.

Thanks!

for your attention

`www.ipipan.waw.pl/~drabent/`

Most of the references to be found in
[D_'18] Drabent, W. Logic + control: On program construction and verification. *Theory and Practice of Logic Programming* 18, 1, 1–29. 2018.

A final version of these slides with contain a reference list

References

K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall. 1997.

[Barták'98] Bartak, R. *Guide to Prolog Programming.* 1998. `http://kti.mff.cuni.cz/~bartak/prolog/`

A. Bossi, N. Cocco. Verifying Correctness of Logic Programs. In *TAPSOFT, Vol.2 (Lecture Notes in Computer Science)*, Vol. 352, J. Díaz, F. Orejas (Eds.), 96–110, Springer, 1989.

Clark, K. L. Predicate logic as computational formalism. Tech. Rep. 79/59, Imperial College, London. December, 1979.

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

[D_+Miłkowska'05] W. Drabent, M. Miłkowska. Proving correctness and completeness of normal programs – a declarative approach. *Theory and Practice of Logic Programming* 5, 6 (2005), 669–711.

[D_'16] Drabent, W. Correctness and completeness of logic programs. *ACM Trans. Comput. Log. 17,* 3, 18:1–18:32. 2016.

[D_'18] Drabent, W. Logic + control: On program construction and verification. *Theory and Practice of Logic Programming* 18, 1, 1–29. 2018.

[D_'20] Drabent, W. The Prolog debugger and declarative programming. In: Gabbrielli M. (eds) *LOPSTR 2019. Lecture Notes in Computer Science*, vol. 12042, Springer. pp. 193-208.

Furia, C. A., Meyer, B., Velder, S. Loop Invariants: Analysis, Classification, and Examples. *ACM Comput. Surv.* 46, 3, Article 34 (January 2014), 51 pages.

C. J. Hogger. *Introduction to Logic Programming.* Academic Press, London, 1984.

Howe, J. M. King, A.. A pearl on SAT and SMT solving in Prolog. *Theor. Comput. Sci. 435*, 43–55. 2012.

R. A. Kowalski. The Relation between Logic Programming and Logic Specification. In *Mathematical Logic and Programming Languages*, C. Hoare, J. Shepherdson (Eds.). Prentice-Hall, 11–27, 1985.
Also in *Phil. Trans. R. Soc. Lond. A*, Vol 312, 1984, 345–361.

Naish, L.: A three-valued declarative debugging scheme. In: *23rd Australasian Computer Science Conference (ACSC 2000).* pp. 166–173. IEEE Computer Society (2000). DOI: 10.1109/ACSC.2000.824398.

Pereira, L.M.: Rational debugging in logic programming. In: Shapiro, E.Y. (ed.) *ICLP'86. Lecture Notes in Computer Science*, vol. 225, pp. 203–210. Springer (1986). Extended version at `https://userweb.fct.unl.pt/~lmp/`

Shapiro, E. *Algorithmic Program Debugging.* The MIT Press, 1983.

L. Sterling and E. Shapiro. *The Art of Prolog* (2 ed.). The MIT Press, 1994.