# Detecting Determinacy in Prolog Programs

Andy King, Lunjin Lu and Samir Genaim

Funded by EP/C015517, INT-0327760 and RS 2005-R3/JP

"you should keep in mind which . . . predicates are determinate, and when they are determinate, and you should provide comments for your own code to remind you of when your own code is determinate" [O'Keefe, 1990]

# Determinacy inference as envisaged at ESOP'05

```
| ?- append([a], [b], L).
L = [a,b] ? ;
no
```

Is append(gr, gr, any) deterministic? ✓

```
| ?- append(X, Y, [a,b]).
X = [], Y = [a,b] ? ;
X = [a], Y = [b] ? ;
X = [a,b], Y = [] ? ;
no
```

Is append(any, any, gr) deterministic? ×

```
| ?- append(X, [b], [a,b]).
X = [a] ? ;
no
```

Is append(any, gr, gr) deterministic? ✓

Or infer append$(x, y, z)$ :- $x \vee (y \wedge z)$ in a **single** application of an analysis

# An answer to a question posed at ESOP'05

- Consider the following program and the results inferred by the ESOP analysis:

| | |
|---|---|
| p(X) :- q(X), r(X). | p($x$) :- $x$ |
| q(a). q(b). | q($x$) :- $x$ |
| r(a). | r($x$) :- *true* |

- These conditions ensure the determinacy of a goal and **all the intermediate sub-goals** invoked whilst solving the goal:
  - Suppose p(X) is the initial goal and X is ground.
  - The leftmost sub-goal q(X) is invoked with X ground and thus deterministic.
  - The rightmost sub-goal r(X) is also invoked with X ground and likewise is deterministic.

# Post-ESOP frustration . . . and an idea

- The goal p(X) generates at most one answer, due to the way the bindings generated by r(X) constrain q(X):

```
| ?- p(X).
X = a ? ;
no
```

- The stronger condition p($x$) :- *true* is also sufficient to ensure that p generates at most one answer at most once
- The goal p(X) illustrates a performance bug: it returns its single answer whilst retaining the choice-point for q(X).
- Ideally the programmer should be alerted to this fault to:
  - add a cut to p(X) to remove the choice-point without compromising correctness;
  - rewrite p(X) so as not to generate any choice-points at all

# Using two forms of determinacy inference for debugging

- Use the old analysis with a new analysis that can detect determinacy in the presence of right-to-left flow of bindings:

| | old analysis | new analysis |
|---|---|---|
| p(X) :- q(X), r(X). | p($x$) :- $x$ | p($x$) :- *true* |
| q(a). q(b). | q($x$) :- $x$ | q($x$) :- $x$ |
| r(a). | r($x$) :- *true* | r($x$) :- *true* |

- Any discrepancy between the results of the two analyses would identify a predicate that is deterministic, yet could possibly leave choice-points on the stack.
- **Just** might provide a way to pinpoint predicates that warrant close scrutiny in third-party software.

# A new analysis for determinacy inference

- Consider the following example:

```
(1) rev([],[]).
(2) rev([X|Xs],Ys) :- rev(Xs,Zs), app(Zs,[X],Ys).

(3) app([],X,X).
(4) app([X|Xs],Ys,[X|Zs]) :- app(Xs,Ys,Zs).
```

- New analysis is comprised of three components:
  1. a component that infers a mutual exclusion condition for each predicate from the success patterns of individual clauses;
  2. a component that enriches that program with delay declarations derived from the mutual exclusion conditions;
  3. a component that applies suspension inference to effect a new form of determinacy inference

# Deriving the mutual exclusion conditions (1 of 2)

- A mutual exclusion condition, when satisfied by a call, ensures that no more than one clause of the matching predicate can lead to a successful derivation.
- Consider characterising the success patterns with an argument-size analysis in which size is measured as list-length:

  (1') rev$(x, y)$ :- $x = 0$, $y = 0$.
  (2') rev$(x, y)$ :- $x \geq 1$, $x = y$.

  (3') app$(x, y, z)$ :- $x = 0$, $y \geq 0$, $y = z$.
  (4') app$(x, y, z)$ :- $x \geq 1$, $y \geq 0$, $x + y = z$.
- Note that one succeed pattern is required for **each clause**.

# Deriving the mutual exclusion conditions (2 of 2)

- Consider a call rev$(x, y)$ where $x$ is bound to a rigid list.
  - Suppose the call succeeds, initially matching clause (1).
  - The presence of the constraint $x = 0$ in success pattern (1') implies that $x$ must be initially bound to [].
  - The constraint $x \geq 1$ in (2') implies that the call cannot also succeed with clause (2).
- Conversely, if the call succeeds by initially matching against clause (2), then the call cannot also succeed with clause (1).
- The rigidity condition $x$ is thus sufficient for mutual exclusion.
- The rigidity of $y$ is also sufficient, hence the combined condition $\mathbf{x} \vee \mathbf{y}$ is also a mutual exclusion condition.
- Repeating this argument for app$(x, y, z)$ yields $\mathbf{x} \vee (\mathbf{y} \wedge \mathbf{z})$.

# Enriching the program with delay declarations

- A call generates at most one answer if each sub-goal satisfies its mutual exclusion condition when it is invoked.
- Suggests considering whether sub-goals can be reordered so they satisfy their mutual exclusion conditions when invoked.
- Delay declarations provide a way to enforce this requirement:

```
delay rev(X, Y) until rigid_list(X); rigid_list(Y).
rev([],[]).
rev([X|Xs],Ys) :- rev(Xs,Zs), app(Zs,[X],Ys).

delay app(X, Y, Z) until rigid_list(X); (rigid_list(Y), rigid_list(Z)).
app([],X,X).
app([X|Xs],Ys,[X|Zs]) :- app(Xs,Ys,Zs).
```

- If a call to the above program does not suspend, then the call to the original program will generate at most one answer.

# Applying suspension inference

- Suspension inference [Genaim and King, ACM ToCL, 2008] can infer calls which cannot lead to suspending states:

$$\begin{aligned} \mathrm{rev}(x, y) &\text{ :- } x \vee y \\ \mathrm{app}(x, y, z) &\text{ :- } x \vee (y \wedge z) \end{aligned}$$

  where $x, y$ and $z$ denote rigidity conditions.

- Using depth-$k$ success pattern analysis gives the following mutual exclusion conditions and transformed code:

$$\begin{aligned} \mathrm{p}(x) &\text{ :- } \mathit{true} \\ \mathrm{q}(x) &\text{ :- } x \\ \mathrm{r}(x) &\text{ :- } \mathit{true} \end{aligned}$$

```
p(X) :- q(X), r(X).
delay q(X) until ground(X).
q(a).    q(b).
r(a).
```

- The result of applying suspension inference is then:

$$\begin{aligned} \mathrm{p}(x) &\text{ :- } \mathit{true} \\ \mathrm{q}(x) &\text{ :- } x \\ \mathrm{r}(x) &\text{ :- } \mathit{true} \end{aligned}$$

  where $x$ denotes a groundness condition.

# Further post-ESOP frustration

- The technique for inferring mutual exclusion conditions is not sensitive to the clause ordering.
- But reasoning about cut should not be **that** tricky:

| | mutual exclusion | new analysis |
|---|---|---|
| p(X, Y) :- q(X), r(X, Y).<br>q(a). q(b).<br>r(X, Y) :- X = a, !, Y = b.<br>r(X, Y) :- ground(X). | p($x, y$) :- *true*<br>q($x$) :- $x$<br>r($x, y$) :- *true* | p($x, y$) :- *true*<br>q($x$) :- $x$<br>r($x, y$) :- *true* |

- Suspension inference infers p(X, Y) :- *true* since q(X), r(X, Y) can be scheduled right-to-left without suspension.
- But the call p(X,Y) has two answers X=a,Y=b and X=b.

# Diagnosing the problem with cut . . . and an idea

- Correctness argument is founded on monotonicity results.
- Cut compromises monotonicity:
  - The call r(X, Y) gives the **single** answer X = a,Y = b.
  - The call r(X, Y) should fail if X = b, but rather succeeds.
- If r(X, Y) is called with X ground then:

| | |
|---|---|
| r(X, Y) :- X = a, !, Y = b.<br>r(X, Y) :- ground(X). | r(X, Y) :- X = a, Y = b.<br>r(X, Y) :- not(X = a), ground(X). |

- Suggests grounding sufficient arguments of a call so that any call arising before a cut is invoked with ground arguments.
- Add groundness requirements to mutual exclusion conditions:

| enriched mutual exclusion | enriched new analysis |
|---|---|
| p($x, y$) :- *true* | p($x, y$) :- $x$ |
| q($x$) :- $x$ | q($x$) :- $x$ |
| r($x, y$) :- $x$ | r($x, y$) :- $x$ |

# Experimental evaluation

| file | time | old | susp | cut |
|---|---|---|---|---|
| aircraft | 2240 | 241 | 241 | +241 |
| asm | 160 | 45 | 45 | +45 |
| boyer | 40 | 33 | 33 | +33 |
| btree | 10 | 10 | +10 | +14 |
| chat-80 | 9710 | 588 | 588 | 588 |
| circuit | 10 | 8 | +9 | 9 |
| conman | 130 | 32 | 32 | +32 |
| cr | 0 | 9 | +9 | 9 |
| cw | 20 | 13 | 13 | +13 |
| dcg | 20 | 21 | +21 | 21 |
| dialog | 20 | 33 | 33 | +33 |
| ili | 220 | 62 | 62 | +63 |

| file | time | old | susp | cut |
|---|---|---|---|---|
| lee-route | 20 | 14 | 14 | +14 |
| nbody | 120 | 48 | 48 | +48 |
| neural | 50 | 50 | +52 | 52 |
| qplan | 230 | 51 | 51 | +52 |
| queens | 20 | 16 | 16 | +16 |
| reducer | 140 | 42 | 42 | +42 |
| robot | 30 | 29 | +29 | +30 |
| sdda | 70 | 33 | +33 | +33 |
| sv | 2280 | 131 | 131 | +131 |
| sa | 120 | 72 | 72 | +72 |
| trs | 3800 | 35 | 35 | +35 |
| tsp | 10 | 23 | +23 | +27 |

# Conclusions

- Cracked the major headaches with determinacy inference
- Experiments suggest analysis is practical and possible useful
- Paper includes extra material on:
  - lemmata that show how determinacy can be observed by observing non-suspension
  - formulates a suspension analysis in terms of a quiescent-state semantics [Saraswat, Rinard, Panangaden, POPL, 1991];
  - tracking inequalities to depth-$k$ is important for precision;
- Future work will exercise the analysis in bug finding