Determinacy Inference for Logic Programs

Lunjin Lu and Andy King

Funded by NSF grants CCR-0131862 and INT-0327760

Outline of this talk

Motivation

Components of our analysis

Experimental evaluation

Conclusions

Motivation

- For a given logic program:
  - instead of *checking* that a *single* goal is deterministic
  - *infer* a *set* of goals that are deterministic in the spirit of [Genaim and Codish, TPLP, 2005; King and Lu, TPLP, 2002]

Motivation

- For a given logic program:
  - instead of *checking* that a *single* goal is deterministic
  - *infer* a *set* of goals that are deterministic in the spirit of [Genaim and Codish, TPLP, 2005; King and Lu, TPLP, 2002]
- "Point-and-click" for easy program development:
  - "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]

Motivation

- For a given logic program:
  - instead of *checking* that a *single* goal is deterministic
  - *infer* a *set* of goals that are deterministic in the spirit of [Genaim and Codish, TPLP, 2005; King and Lu, TPLP, 2002]
- "Point-and-click" for easy program development:
  - "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]
- Binding-time analysis for inferring unfolding conditions for a non-leftmost call [Leuschel *et al*, TPLP, 2004]

Motivation

- For a given logic program:
  - instead of *checking* that a *single* goal is deterministic
  - *infer* a *set* of goals that are deterministic in the spirit of [Genaim and Codish, TPLP, 2005; King and Lu, TPLP, 2002]
- "Point-and-click" for easy program development:
  - "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]
- Binding-time analysis for inferring unfolding conditions for a non-leftmost call [Leuschel *et al*, TPLP, 2004]
- Some anti-motivation: [Mogensen, Ershov Memorial, 1996; Hermenegildo *et al*, LOPSTR, 2004/05; *etc*]

Appending one list to another

(1) append([], Ys, Ys).
(2) append([X|Xs], Ys, [X|Zs]) :- append(Xs, Ys, Zs).

| ?- 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

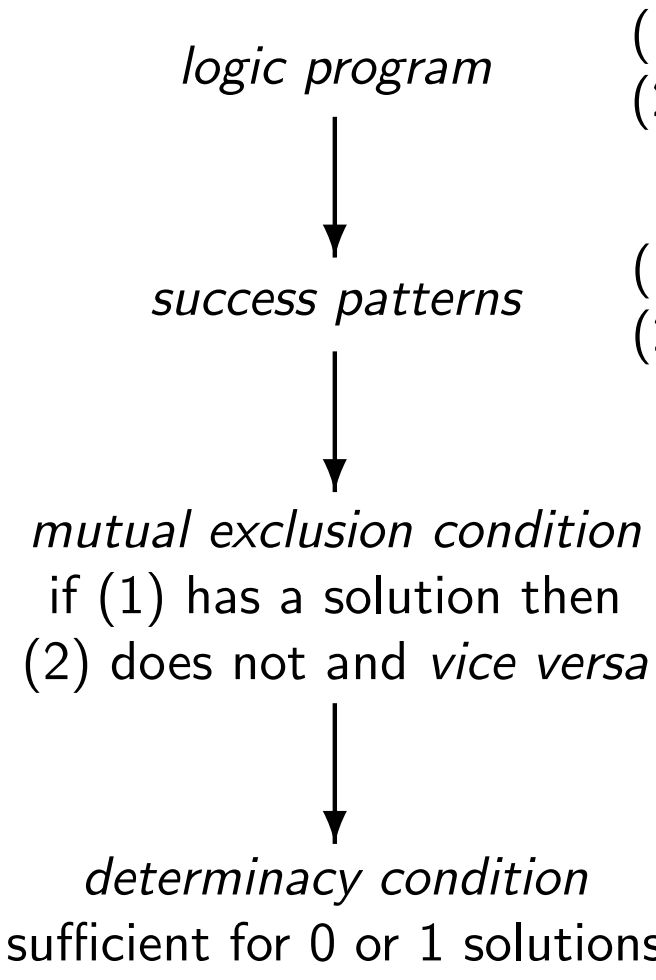
Is append(any, any, gr) deterministic? ×

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

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

Instead infer append(gr, any, any) *and* append(any, gr, gr) are deterministic

Determinacy inference architecture



(1) append([], Ys, Ys).
(2) append([X|Xs], Ys, [X|Zs]) :- append(Xs, Ys, Zs).

(1) append(x_1, x_2, x_3) :- $x_1 = 0 \wedge x_2 \geq 0 \wedge x_2 = x_3$.
(2) append(x_1, x_2, x_3) :- $x_1 \geq 1 \wedge x_2 \geq 0 \wedge x_1 + x_2 = x_3$.

append(x_1, x_2, x_3) :- $x_1 \vee (x_2 \wedge x_3)$.

append(x_1, x_2, x_3) :- $x_1 \vee (x_2 \wedge x_3)$.

Interpreting success patterns

- The following success patterns describe the sizes of the arguments in any solution to a query:

  (1) append(x_1, x_2, x_3) :- $x_1 = 0 \wedge x_2 \geq 0 \wedge x_2 = x_3$.
  (2) append(x_1, x_2, x_3) :- $x_1 \geq 1 \wedge x_2 \geq 0 \wedge x_1 + x_2 = x_3$.
- Success pattern (1) says
  - if clause (1) is initially used to resolve append(t_1, t_2, t_3) *and*
  - θ is a computed answer substitution *then*
  - $\theta(t_1)$ is a list with zero elements
- Success pattern (2) says
  - if clause (2) is initially used to resolve append(t_1, t_2, t_3) *and*
  - θ is a computed answer *then*
  - $\theta(t_1)$ is a list with at least one element
- Determinancy inference is applicable with other notions of size and other types of (standard) success pattern analysis

Inferring success patterns

This slide is intentionally left blank

Calculating mutual exclusion conditions

- Consider append(any, gr, gr) so the 2nd and 3rd arguments are unchanged during the execution of the call:
  - (1) append(x_1, x_2, x_3) :- c_1 $\quad c_1 = x_1 = 0 \wedge x_2 \geq 0 \wedge x_2 = x_3$.
    (2) append(x_1, x_2, x_3) :- c_2 $\quad c_2 = x_1 \geq 1 \wedge x_2 \geq 0 \wedge x_1 + x_2 = x_3$.
- If clause (1) leads to a solution, then the relative sizes of 2nd and 3rd arguments after and before the call are given by:
  - $\exists_{x_1}(c_1) = (x_2 \geq 0 \wedge x_2 = x_3)$
- As these call arguments satisfy $\exists_{x_1}(c_1)$ they cannot satisfy:
  - $\exists_{x_1}(c_2) = (x_2 \geq 0 \wedge 1 + x_2 \leq x_3)$

  therefore clause (2) cannot also give a solution.
- Hence, for append(any, gr, gr), if clause (1) leads to a solution then clause (2) cannot and *vice versa*.
- One mutual exclusion condition is append(x_1, x_2, x_3) :- $x_2 \wedge x_3$

Strategy for mutual exclusion conditions

- Let $X = \{x_1, x_2, x_3\}$
- Construct append(x_1, x_2, x_3) :- f where:

$$f = \vee \left\{ \wedge Y \;\middle|\; \begin{array}{l} Y \subseteq X \\ \exists_{X \setminus Y}(c_1) \wedge \exists_{X \setminus Y}(c_2) \text{ is unsatisfiable} \end{array} \right\}$$

- Gives append(x_1, x_2, x_3) :- $x_1 \vee (x_2 \wedge x_3)$
- Need to be careful for larger arity predicates

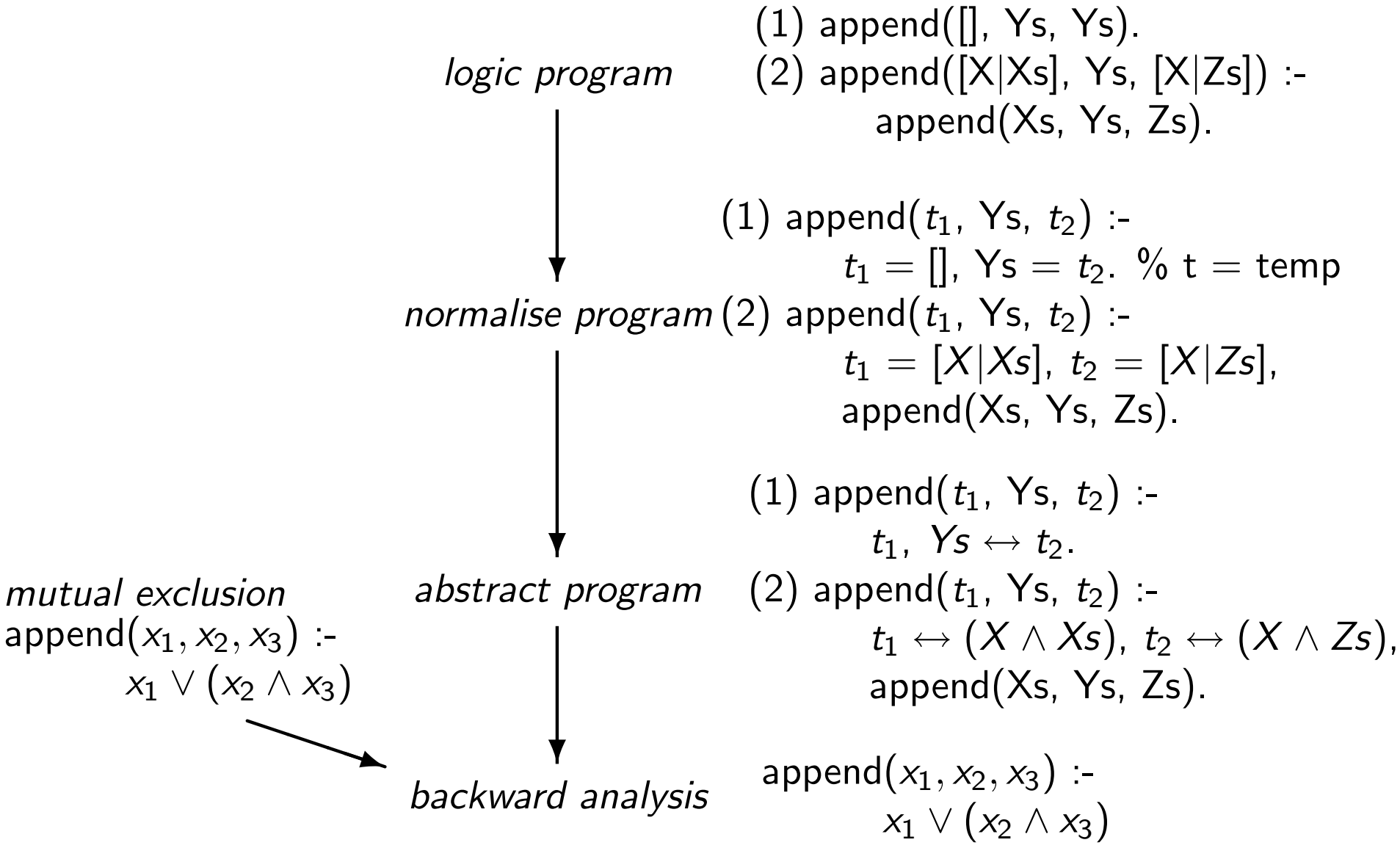
Calculating determinacy conditions

- A query is deterministic if each call encountered whilst solving the query satisfies its mutual exclusion condition
- For example, consider the query append(any, gr, gr) where:

  (1) append([], Ys, Ys).
  (2) append([X|Xs], Ys, [X|Zs]) :- append(Xs, Ys, Zs).

- The top-level query append(any, gr, gr) satisfies the mutual exclusion condition
- If clause (2) is selected, then Ys and [X|Zs] are grounded, hence append(Xs, Ys, Zs) also also satisfies its mutual exclusion condition
- But this is determinacy checking not determinacy inference

Component for inferring determinacy requirements



One part of the backward analysis

Consider the abstract version of the program:

(1) append(t_1, Ys, t_2) :- t_1, $Ys \leftrightarrow t_2$.
(2) append(t_1, Ys, t_2) :-
$t_1 \leftrightarrow (X \wedge Xs)$, $t_2 \leftrightarrow (X \wedge Zs)$, append(Xs, Ys, Zs).
$\uparrow f_1$ $\uparrow f_2$ $\uparrow f_3$

To infer a determinancy condition for clause (2):

- Know $f_3 = Xs \vee (Ys \wedge Zs)$
- Compute $f_2 = (t_2 \leftrightarrow (X \wedge Zs)) \rightarrow f_3$
- Compute $f_1 = (t_1 \leftrightarrow (X \wedge Xs)) \rightarrow f_2$
- Require a condition f expressed in terms of t_1, Ys, t_2 such that f_1 holds if f holds — calculate $f = \forall_X(\forall_{Xs}(\forall_{Zs}(f_1))) = t_1 \vee (Ys \wedge t_2)$

A determinacy condition for whole predicate is thus:

- $true \wedge (x_1 \vee (x_2 \wedge x_3)) \wedge (x_1 \vee (x_2 \wedge x_3)) = x_1 \vee (x_2 \wedge x_3)$

Experimental evaluation

benchmark	*predicate*	*exclusion cond*	*determinacy cond*
treesort – disjunctive conditions for multi-mode predicates	`tree_to_list_aux`(x_1, x_2, x_3)	x_1	x_1
	`tree_to_list`(x_1, x_2)	*true*	x_1
	`list_to_tree`(x_1, x_2)	*true*	x_1
	`insert_list`(x_1, x_2, x_3)	x_1	$x_1 \wedge x_2$
	`insert`(x_1, x_2, x_3)	$x_1 \wedge (x_2 \vee x_3)$	$\underline{x_1 \wedge (x_2 \vee x_3)}$
	`treesort`(x_1, x_2)	*true*	x_1
queens – *false* for non-determinate predicates	`noattack`(x_1, x_2, x_3)	x_2	x_2
	`safe`(x_1)	x_1	x_1
	`delete`(x_1, x_2, x_3)	*false*	*false*
	`perm`(x_1, x_2)	$x_1 \vee x_2$	$\underline{\textit{false}}$
	`queens`(x_1, x_2)	*true*	*false*
permsort – exclusion conditions interesting	`select`(x_1, x_2, x_3)	$\underline{\textit{false}}$	*false*
	`ordered`(x_1)	x_1	x_1
	`perm`(x_1, x_2)	$x_1 \vee x_2$	*false*
	`sort`(x_1, x_2)	*true*	*false*
serialize – susceptable to false positives	`arrange0`(x_1, x_2)	$x_1 \vee x_2$	x_1
	`numbered`(x_1, x_2, x_3)	x_1	x_1
	`palin`(x_1)	*true*	*true*
	`pairlists`(x_1, x_2, x_3)	$x_1 \vee x_2 \vee x_3$	$x_1 \vee x_2 \vee x_3$
	`serialize0`(x_1, x_2)	*true*	$x_1 \wedge x_2$
	`split0`(x_1, x_2, x_3, x_4)	$x_1 \wedge x_2$	$x_1 \wedge x_2$
	`go`(x_1)	*true*	$\underline{\textit{false}}$

Performance evaluation

	argument-size			*depth-k*		
file	*succ*	*lfp*	*gfp*	*succ*	*lfp*	*gfp*
boyer	1666	591	60	441	360	50
bryant	7522	261	90	371	321	80
chat_80	67393	2153	431	494578	4977	631
ga	2146	161	40	2814	290	40
ili	2314	450	111	1222	531	100
nand	17921	1682	421	841	801	260
nbody	877	191	30	241	301	80
peep	404	170	30	761	441	70
peval	6938	621	100	4466	611	90
press	584	251	40	320	381	70
reducer	7867	270	50	190	301	50
rubik	30150	420	70	571	3755	221
sim	10270	561	171	35411	611	100
sim_v5-2	2948	581	170	491	701	180
trs	13174	280	91	321	450	100

Conclusions

- Can infer sufficient conditions for determinacy (rather than just check determinacy)
- Determinacy inference analysis is composed of off-the-self success pattern analysis augmented with techniques for:
  - synthesising mutual exclusion conditions
  - backwards analysis for groundness *or* rigidity
- Initial experiments suggest the analysis is practical and useful