

(Logic) Program Specialisation

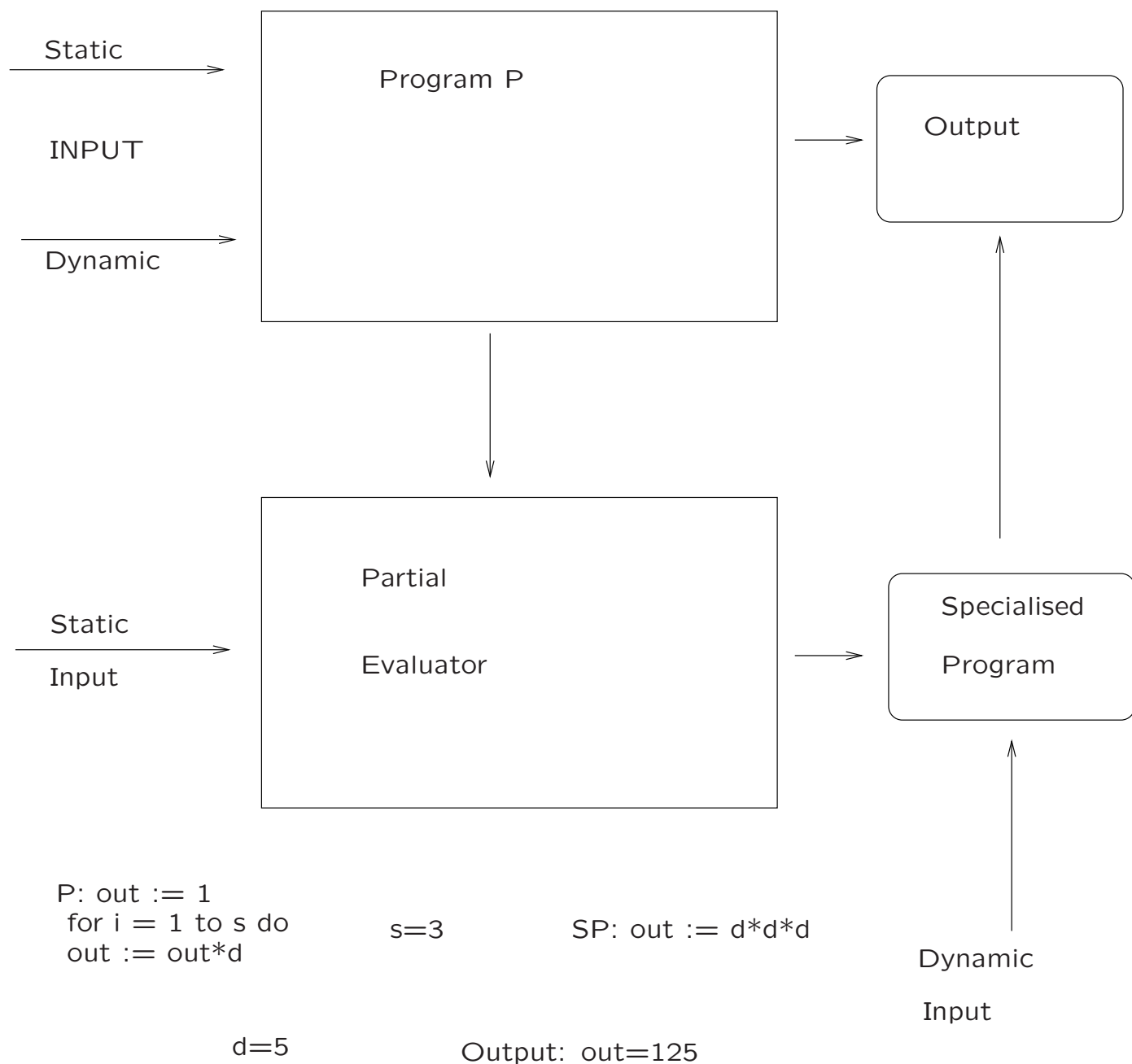
more material in:

M. Leuschel and M. Bruynooghe, Logic program specialisation through partial deduction: Control issues, *Theory and Practice of Logic Programming* **2(4&5)**, 2002, pp 461–515.

- Source to source transformation
- Improve performance of source program
 - exploitation of partial input
 - elimination of inefficiencies
 - preservation of semantics
- support for simple, easily maintainable code
- support for overly general code
- facilitated by clear and simple semantics

Partial Evaluation

Specialisation of function: S-M-N theorem (Kleene 1952)



Partial Deduction

No matter how little input, SLDNF-tree can always be constructed

But maybe more branches, perhaps infinite

Basic idea:

- construct partial (finite) SLDNF-tree (“un-folding”) for query
- extract resultants from non-failing branches
- if needed, construct more partial SLDNF-trees
- the set of resultants is the new program
- post-processing

Partial SLD-derivation with accumulated substitution θ :

$\leftarrow A$

\vdots

$\leftarrow B_1, \dots, B_m$

The resultant is

$A\theta \leftarrow B_1, \dots, B_m$

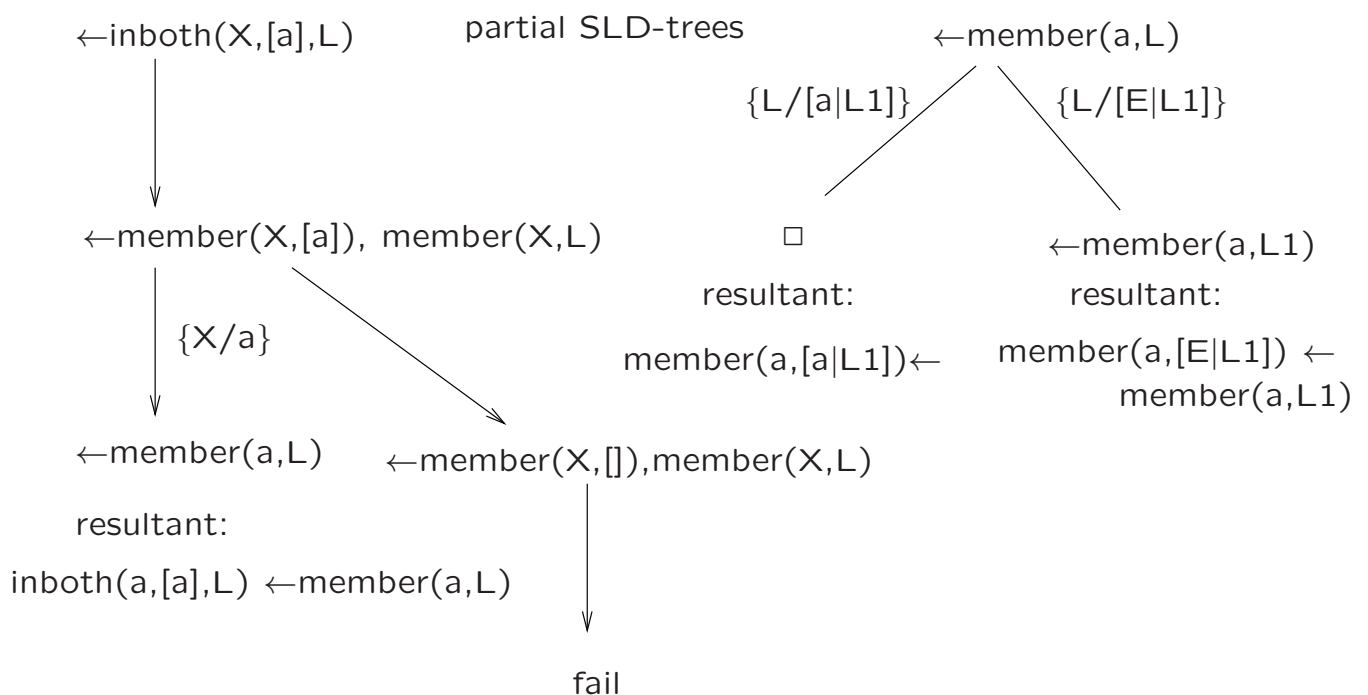
example

$\text{member}(X, [X|T]) \leftarrow.$

$\text{member}(X, [Y|T]) \leftarrow \text{member}(X, T).$

$\text{inboth}(X, L1, L2) \leftarrow \text{member}(X, L1), \text{member}(X, L2).$

query: $\leftarrow \text{inboth}(X, [a], L).$



$\text{inboth}(a, [a], L) \leftarrow \text{member}(a, L).$

$\text{member}(a, [a|L1]) \leftarrow.$

$\text{member}(a, [E|L1]) \leftarrow \text{member}(a, L1).$

postprocessing: argument filtering

query: preserve only its variables X and L :

define $ib(X, L) \leftarrow inboth(X, [a], L)$.

built tree for $\leftarrow ib(X, L)$

resultant becomes: $ib(a, L) \leftarrow member(a, L)$.

$member(a, L)$: preserve only variables shared with rest of resultants:

define $mb(L) \leftarrow member(a, L)$.

built tree for $\leftarrow mb(L)$

resultants: $mb([a|L1]) \leftarrow$.

and $mb([E|L1]) \leftarrow member(a, L1)$.

Finally *fold* the occurrences of $member(a, L)$ in bodies. (To be correct, unfolding has to give back the original)

Final program:

$ib(a, L) \leftarrow mb(L)$.

$mb([a|L1]) \leftarrow$.

$mb([E|L1]) \leftarrow mb(L1)$.

when equivalent? Lloyd-Shepherdson

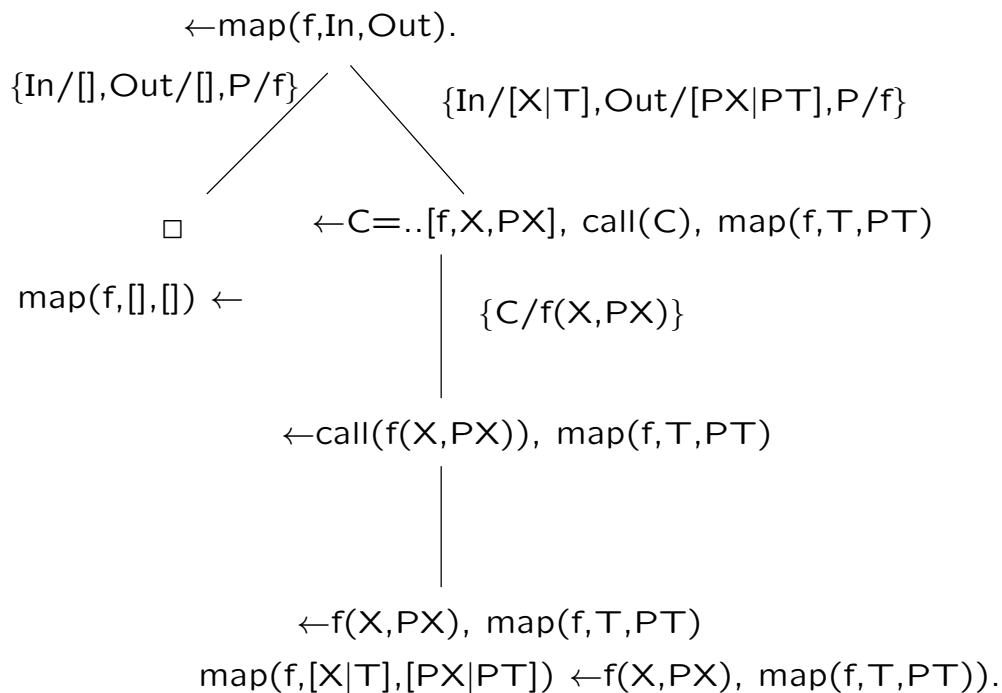
- Trees not trivial, more than one node
trivial tree: resultant is $p \leftarrow p$
- *CLOSED*: atoms in the leaves are instances of root atoms
otherwise: missing branch, missing solution possible
- *INDEPENDANCE*: root atoms do not have common instances
otherwise: redundant solutions and more instantiated cas. possible,

Given: program P , set of atoms \mathcal{A} , for each A in \mathcal{A} a non-trivial partial SLD-tree for the query $\leftarrow A$. Let P' be the set of resultants. If P' is \mathcal{A} -closed i.e. all atoms in P' are instances of atoms in \mathcal{A} the heads are instances by construction and the elements of \mathcal{A} are independent, then

- $P' \cup \{\leftarrow A\}$ has a refutation with cas θ iff $P \cup \{\leftarrow A\}$ does.
- $P' \cup \{\leftarrow A\}$ has a finitely failed SLDNF-tree iff $P \cup \{\leftarrow A\}$ does.

Example

```
map(P, [], []) ←.
map(P, [X|T], [PX|PT]) ←
    C=..[P,X,PX], call(C), map(P, T, PT).
call(C) ← C.
```



```
map(f, [], []) ←.
map(f, [X|T], [PX|PT]) ←
    f(X, PX), map(f, T, PT).
```

Code for f/2 is left untouched one step unfolding
of f(X,Y): resultants are original clauses
overhead higher order is eliminated
postprocessing can eliminate first argument

Which atoms in \mathcal{A} ?

When to stop unfolding?

CONTROL

GLOBAL control: choice of \mathcal{A} , on which atoms is partial deduction applied

LOCAL control: how the SLDNF-trees look like

- Correctness: Local: no trivial trees
Global: closedness, independence
- Termination: Local: finite tree
Global: finite set \mathcal{A}
- Precision: Local: which atoms to unfold, how far?
Global: which atoms in \mathcal{A} , more general than required for closedness?

A naive algorithm for specialising $\leftarrow A$:

- Construct tree for $\leftarrow A$, i.e. unfold as long as it looks interesting
- WHILE there is an atom B in a leaf which is not an instance of a root of a tree
DO: construct a tree for $\leftarrow B$

Problems:

independence : several roots may have common instances

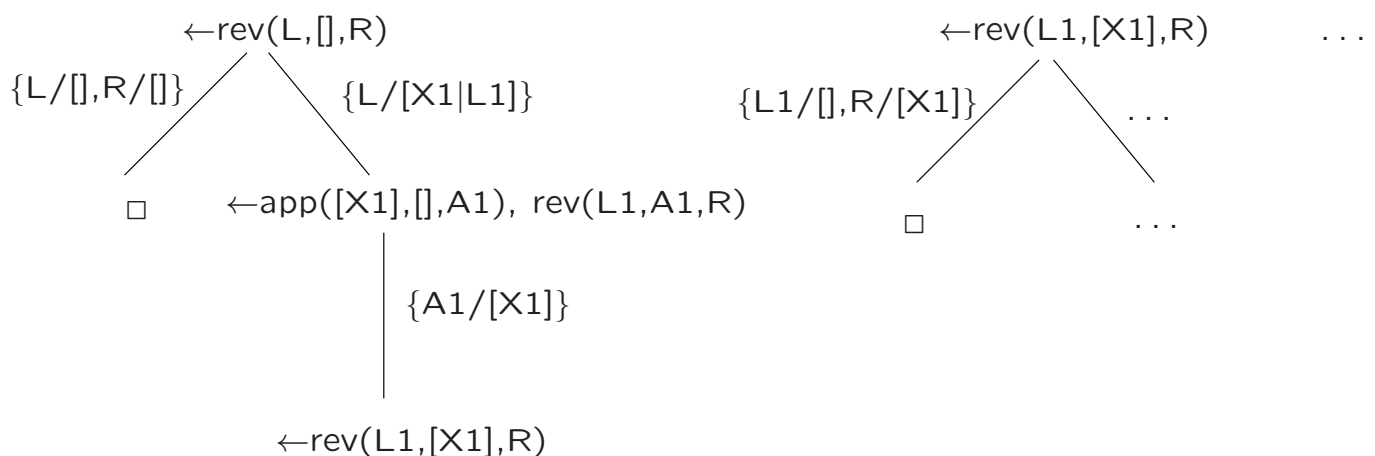
global termination: one may end up with infinite set of trees

$\text{rev}([],L,L) \leftarrow .$

$\text{rev}([X|Xs],Acc,R) \leftarrow$

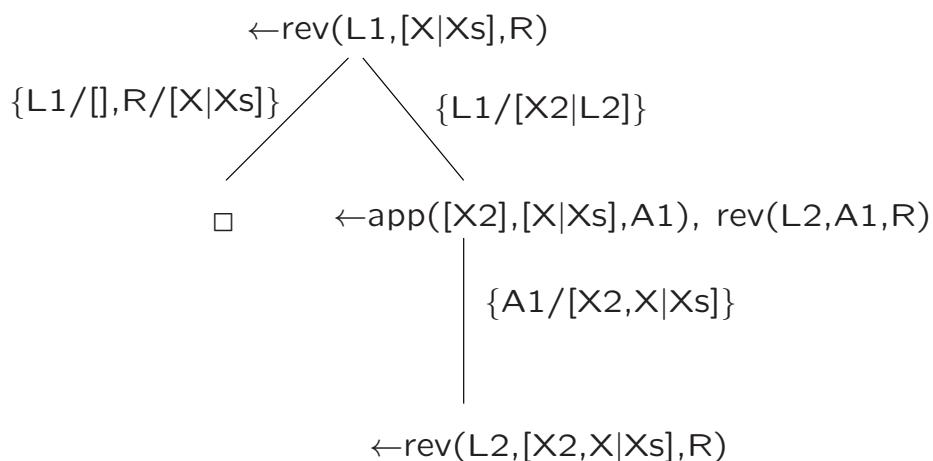
$\text{app}([X],Acc,Nacc), \text{reverse}(Xs,Nacc,R).$

$\leftarrow \text{rev}(L,[],R).$



Solution for **global termination:** *generalisation*

- start over and built tree for least general generalisation of $\text{rev}(L, [], R)$ and $\text{rev}(L, [X], R)$ which is $\text{rev}(L, \text{Acc}, R)$: all input lost
- keep first tree and construct tree for least general generalisation of $\text{rev}(L, [X1], R)$ and $\text{rev}(L, [X1, X2], R)$ which is $\text{rev}(L, [X|Xs], R)$: some input preserved



Generalisation to be balanced against precision

Solution for **independence**:

renaming is often better than generalisation

$\text{member}(a, L)$ and $\text{member}(X, [b])$ are dependent (common instance $\text{member}(a, [b])$)

Their generalisation: $\text{member}(X, L)$: no specialisation

Better: define $\text{member1}(X) \leftarrow \text{member}(X, [b])$
and $\text{member2}(L) \leftarrow \text{member}(a, L)$

turn tree for $\leftarrow \text{member}(X, [b])$ into tree for $\leftarrow \text{member1}(X)$

and tree for $\leftarrow \text{member}(a, L)$ into tree for $\leftarrow \text{member2}(L)$

fold leaves:

$\text{member}(a, L)$ is folded into $\text{member2}(L)$

$\text{member}(X, [b])$ is folded into $\text{member1}(X)$ and

$\text{member}(a, [b])$ in either $\text{member1}(a)$ or $\text{member2}([b])$

(if one did not decide to specialise it separately)

Local termination

when to stop unfolding

1. determinacy

- Only unfold atoms which match at most one clause head
- + lookahead: if all but one branch “soon” fail
- To avoid trivial trees: allow that first step is nondet unfolding of leftmost atom
- To get more unfolding: allow one nondet step
- risk nontermination
- risk: nondet unfolding of atom which is not first can cause serious slowdown

$\leftarrow \text{big}(\dots), p(\dots), \text{more.}$

$p(\dots) \leftarrow q1(\dots).$

$p(\dots) \leftarrow q2(\dots).$

unfolding p gives resultants of form

$\dots \leftarrow \text{big}(\dots), q1(\dots), \text{more.}$

$\dots \leftarrow \text{big}(\dots), q2(\dots), \text{more.}$

$\text{big}(\dots)$ is recomputed on backtracking,
was computed once in original program

Local termination (cont)

2. Well-founded order

Unfold as long a successive nodes of derivation can be mapped to a decreasing sequence of a well-founded set (a partial order not allowing an infinite decreasing sequence)

intuition: unfold as long as size of some “input” decreases

Wellfounded order has either

$p([], [a]) > p([a], [])$ or

$p([a], []) > p([], [a])$

lots of refinements: not comparing with immediate predecessor

Local termination (cont)

3. Well-quasi order

A quasi order (reflexive, transitive), which allows no infinite admissible sequences

Admissible sequence: $\dots, s_i, \dots, s_j, \dots$ for which no $i < j$ such that $s_i \leq s_j$

Unfold as long as successive nodes are admissible sequence

Need to compare with every predecessor

Homeomorphic embedding

$X \sqsubseteq Y$ for every pair of variables

$s \sqsubseteq f(t_1, \dots, t_n)$ if $s \sqsubseteq t_i$ for some i

$f(s_1, \dots, s_n) \sqsubseteq f(t_1, \dots, t_n)$ if $s_i \sqsubseteq t_i$ for all i

intuition $A \sqsubseteq B$ if A obtainable from B by striking out symbols

$p([], [a])$ not embedded in $p([a], [])$ and $p([a], [])$ not embedded in $p([], [a])$

algorithm

Given program P and goal $\leftarrow A$

Init: $i := 0$, $\mathcal{A}_0 := \{A\}$

repeat:

 for each A in \mathcal{A}_i construct tree for $\leftarrow A$
 using unfolding rule \mathcal{U}

$\mathcal{A}'_i = \mathcal{A}_i \cup \{\text{leaves}\}$

$\mathcal{A}_{i+1} := \text{abstract}(\mathcal{A}'_i)$

$i := i + 1$

until $\mathcal{A}_{i+1} = \mathcal{A}_i$

apply renaming to ensure independence

construct P' by taking resultants

abstraction: balance between ensuring termination and maximising specialisation

abstraction controls polyvariance

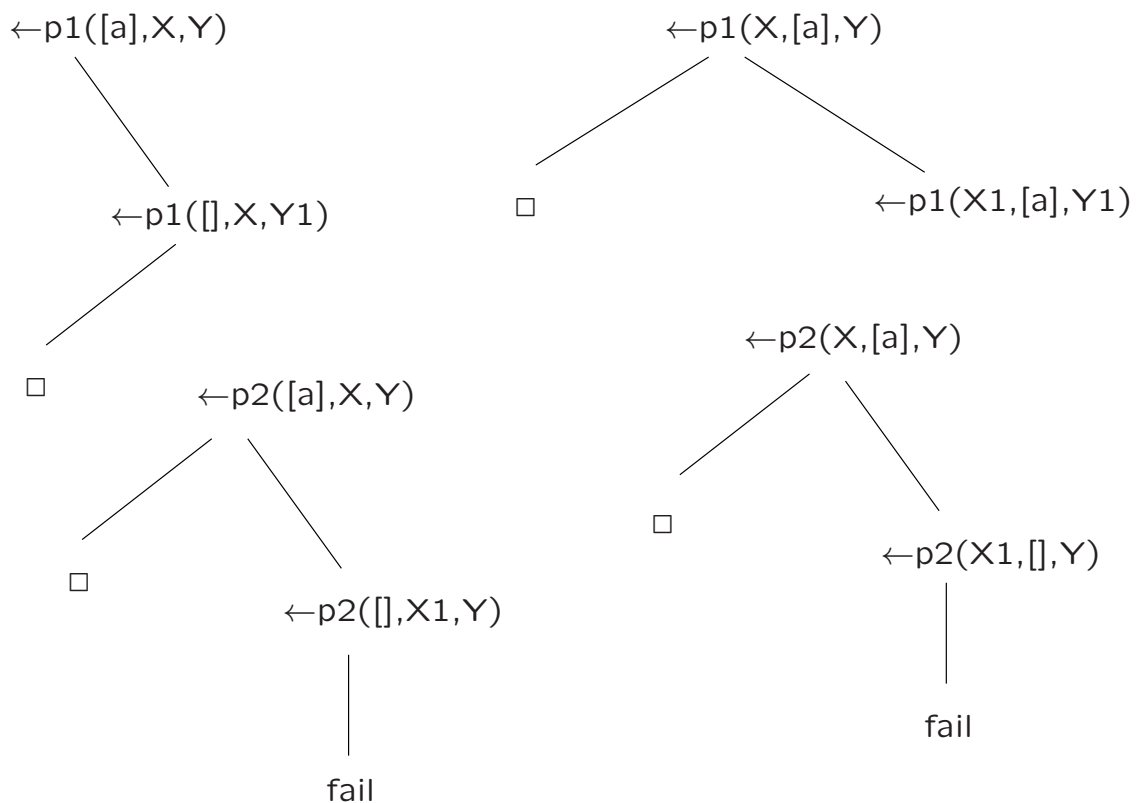
A simple abstraction: partition atoms based on syntactic structure, replace each set by its least general generalisation

$p1([],X,X) \leftarrow.$

$p1([X|U],V,[X|W]) \leftarrow p1(U,V,W).$

$p2([X|Xs],[X|Ys],X) \leftarrow.$

$p2([X|Xs],[Y|Ys],R) \leftarrow p2(Xs,Ys,R).$



calls to p1 very different

calls to p2: same resultant:

$p2([a|Xs],[a|Ys],a) \leftarrow.$

Characteristic trees

Leuschel: Use of characteristic tree to partition atoms

Char. tree: set of branches

branch: sequence of nodes

node: litnr - clausnr

tree for $p1([a],X,Y)$: $\{ \langle 1-2, 1-1 \rangle \}$

tree for $p1(X,[a],Y)$: $\{ \langle 1-1 \rangle, \langle 1-2 \rangle \}$

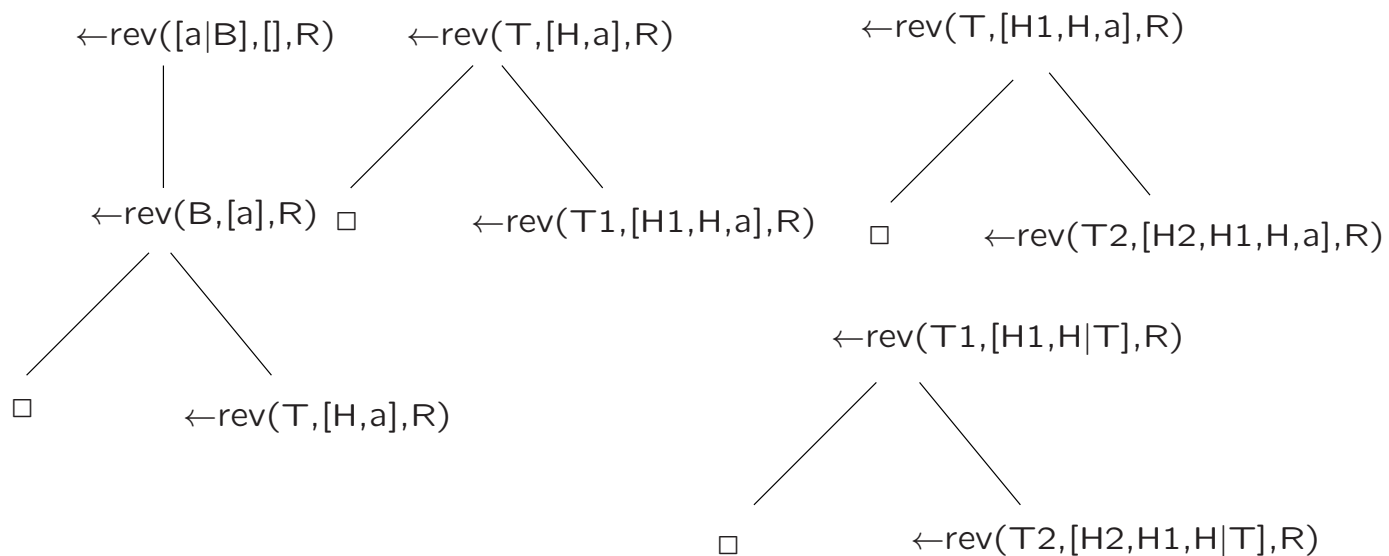
tree for $p2([a],X,Y)$: $\{ \langle 1-1 \rangle \}$

tree for $p2(X,[a],Y)$: $\{ \langle 1-1 \rangle \}$

$rev([], Acc, Acc) \leftarrow$.

$rev([H|T], Acc, Res) \leftarrow rev(T, [H|Acc], Res)$.

unfolding: homeomorphic embedding



BUT generalised atom may have larger characteristic tree
enforce same tree
take care to avoid use of code for too general calls

BUT Risk of nontermination:
infinite number of different atoms with different characteristic tree is possible
arbitrary depth bound never good
Organise atoms to be unfolded in dependency tree
use homeomorphic embedding to control growth of that tree

ECCE system of Leuschel

Conjunctive Partial Deduction

Conjunctions of atoms as root of tree:

Power of *fold/unfold* transformations

Tupling: input is consumed twice

$ml(L, M, N) \leftarrow mx(L, M), ln(L, N).$

$mx([], 0) \leftarrow.$

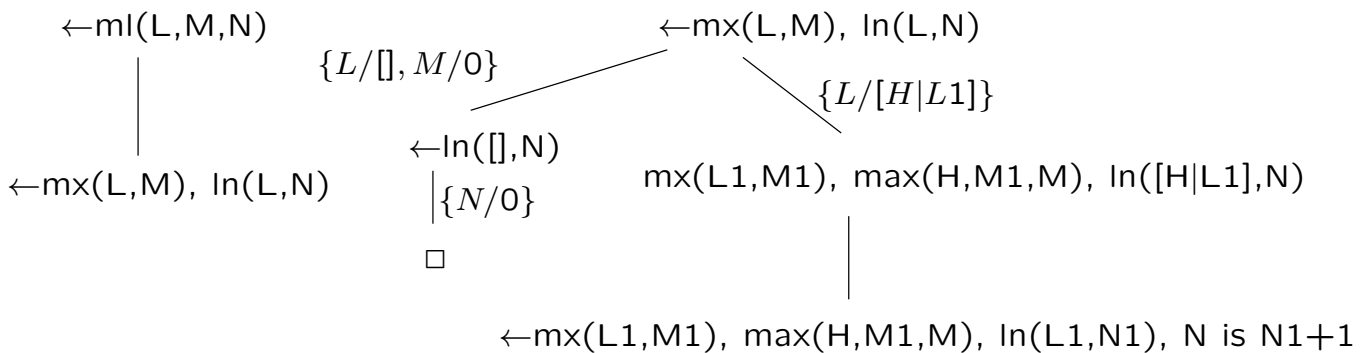
$mx([H|T], R) \leftarrow mx(T, M), max(H, M, R).$

$max(M, N, M) \leftarrow M > N.$

$max(M, N, N) \leftarrow M \leq N.$

$ln([], 0) \leftarrow.$

$ln([H|T], R) \leftarrow ln(T, N), R \text{ is } N+1.$



Define $mxl(L, M, N) \leftarrow mx(L, M), ln(L, N).$,

extend second tree with root $\leftarrow mxl(L, M, N),$

fold leaves and extract resultants:

$ml(L, M, N) \leftarrow mxl(L, M, N).$

$mxl([], 0, 0) \leftarrow.$

$mxl([H|L1], M, N) \leftarrow mxl(L1, M1, N1), max(H, M1, M),$
 $N \text{ is } N1+1.$

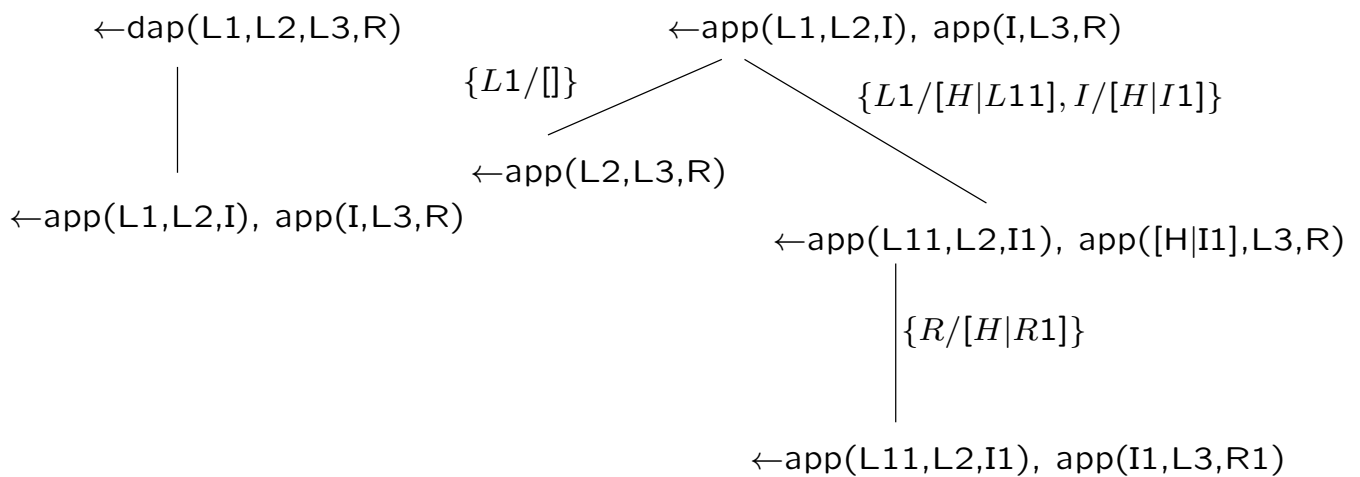
Deforestation:

elimination of intermediate data structure

$dapp(L1, L2, L3, R) \leftarrow app(L1, L2, I), app(I, L3, R).$

$app([], L, L) \leftarrow.$

$app([X|L1], L2, [X|R]) \leftarrow app(L1, L2, R).$



Define $appapp(L1, L2, L3, R) \leftarrow app(L1, L2, I), app(I, L3, R).$

I can be dropped as it is everywhere local to the conjunction

The second tree can be extended into a tree for $\leftarrow appapp(L1, L2, L3, R).$

Fold leaves and extract resultants:

(fold is reversible, dropping more arguments in $appapp$ would make it irreversible!)

$dapp(L1, L2, L3, R) \leftarrow appapp(L1, L2, L3, R).$

$appapp([], L2, L3, R) :- app(L2, L3, R).$

$appapp([H|L11], L2, L3, [H|R1]) \leftarrow$
 $appapp(L11, L2, L3, R1).$

combine with abstract interpretation

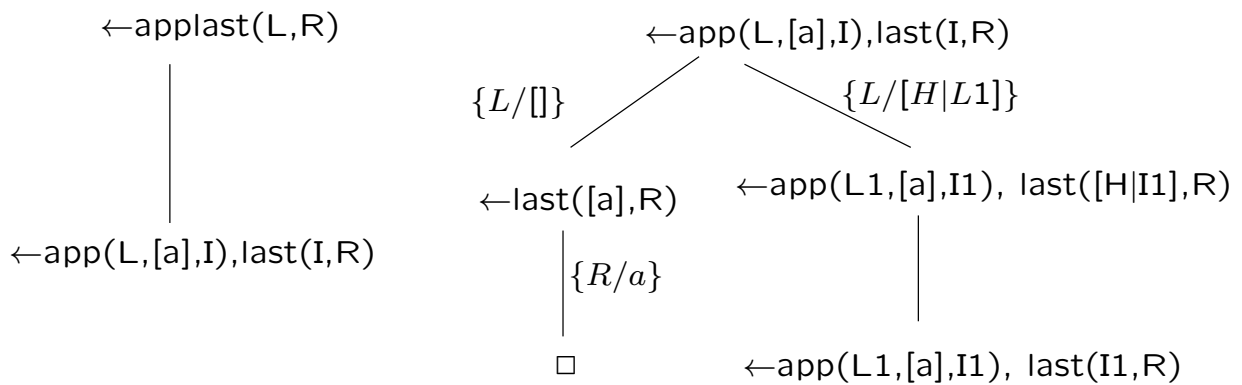
$\text{applast}(L,R) \leftarrow \text{app}(L,[a],I),\text{last}(I,R).$

$\text{app}([],L,L) \leftarrow.$

$\text{app}([X|L1],L2,[X|R]) \leftarrow \text{app}(L1,L2,R).$

$\text{last}([X],X) \leftarrow.$

$\text{last}([H|T],R) \leftarrow \text{last}(T,R).$



Define $al(L,R) \leftarrow \text{app}(L,[a],I),\text{last}(I,R)$

extend second tree in tree for $\leftarrow al(L,R)$

fold and extract resultants

$\text{applast}(L,R) \leftarrow al(L,R).$

$al([],a) \leftarrow.$

$al([H|L1],R) \leftarrow al(L1,R).$

Abstract nonground successset of $al/4$ predicate

least general generalisation as abstraction operator

first iteration: $al([],a)$

second: $\alpha(\{al([],a),al([H],a)\}) = al(L,a)$

third: $\alpha(\{al([],a),al([H1|L],a)\}) = al(L,a)$ (fixpoint)

apply $mgu(atom,success\ set)$ on all clauses with an atom al :

$applast(L,a) \leftarrow al(L,a).$

$al([],a) \leftarrow.$

$al([H|L1],a) \leftarrow al(L1,a).$

Define $list(L) \leftarrow al(L,a),$

Build tree for $List(L)$, fold leaves al with new definition (argument filtering):

$applast(L,a) \leftarrow list(L).$

$list([]) \leftarrow.$

$list([H|L1]) \leftarrow list(L1).$

This was *online* partial deduction: unfolding decisions and construction of trees (code generation) in same phase

offline partial deduction:

phase I: make decisions about unfolding, yields annotated program (form of abstract interpretation)

phase II: unfold and generate code

popular for partial evaluation of functional programming: no unification but matching (one way information passing)

annotate to unfold if input available

e.g. append: if first argument available

Mercury: adds types + modes, replaces unification by matching

offline partial deduction feasible

Exercise

Consider the program:

$\text{sumandproduct}(L,S,P) \leftarrow \text{sum}(L,0,S), \text{product}(L,1,P).$

$\text{sum}([],\text{Acc},\text{Acc}) \leftarrow.$

$\text{sum}([X|L],\text{PS},S) \leftarrow \text{PS1 is PS}+X, \text{sum}(L,\text{PS1},S).$

$\text{product}([],\text{Acc},\text{Acc}) \leftarrow.$

$\text{product}([X|L],\text{PP},P) \leftarrow \text{PP1 is PP} * X, \text{product}(L,\text{PP1},P).$

Use conjunctive partial deduction to specialise this program for the query $\leftarrow \text{sumandproduct}([2|L],S,P).$