Derivation of Efficient Logic Programs by Specialization and Reduction of Nondeterminism

Maurizio Proietti — IASI-CNR, Rome, Italy

Sophie Renault — European Patent Office, Rijswijk, The Netherlands

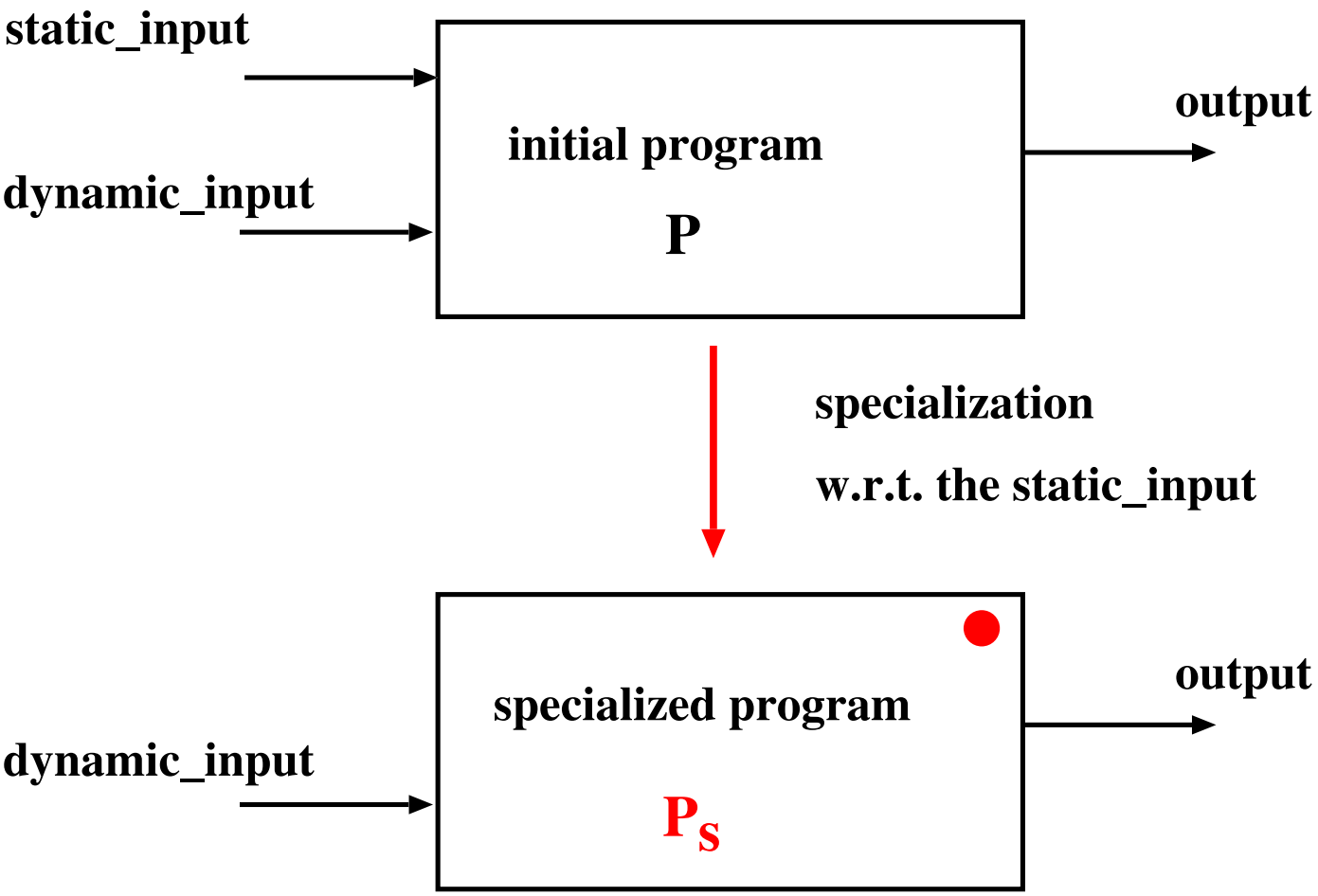
Alberto Pettorossi — University of Roma Tor Vergata, Rome, Italy

WRSC2003 - Workshop on Robust Software Construction

Hayama, Kanagawa, Japan

February 28th - March 2nd, 2003

Program Specialization



For logic programs it is called Partial Deduction

Program Specialization :

program improvement when knowing the context of use

e.g., SORTING a list when ordered $\rightarrow$ IDENTITY

eliminating unnecessary data structures

getting tail recursive programs

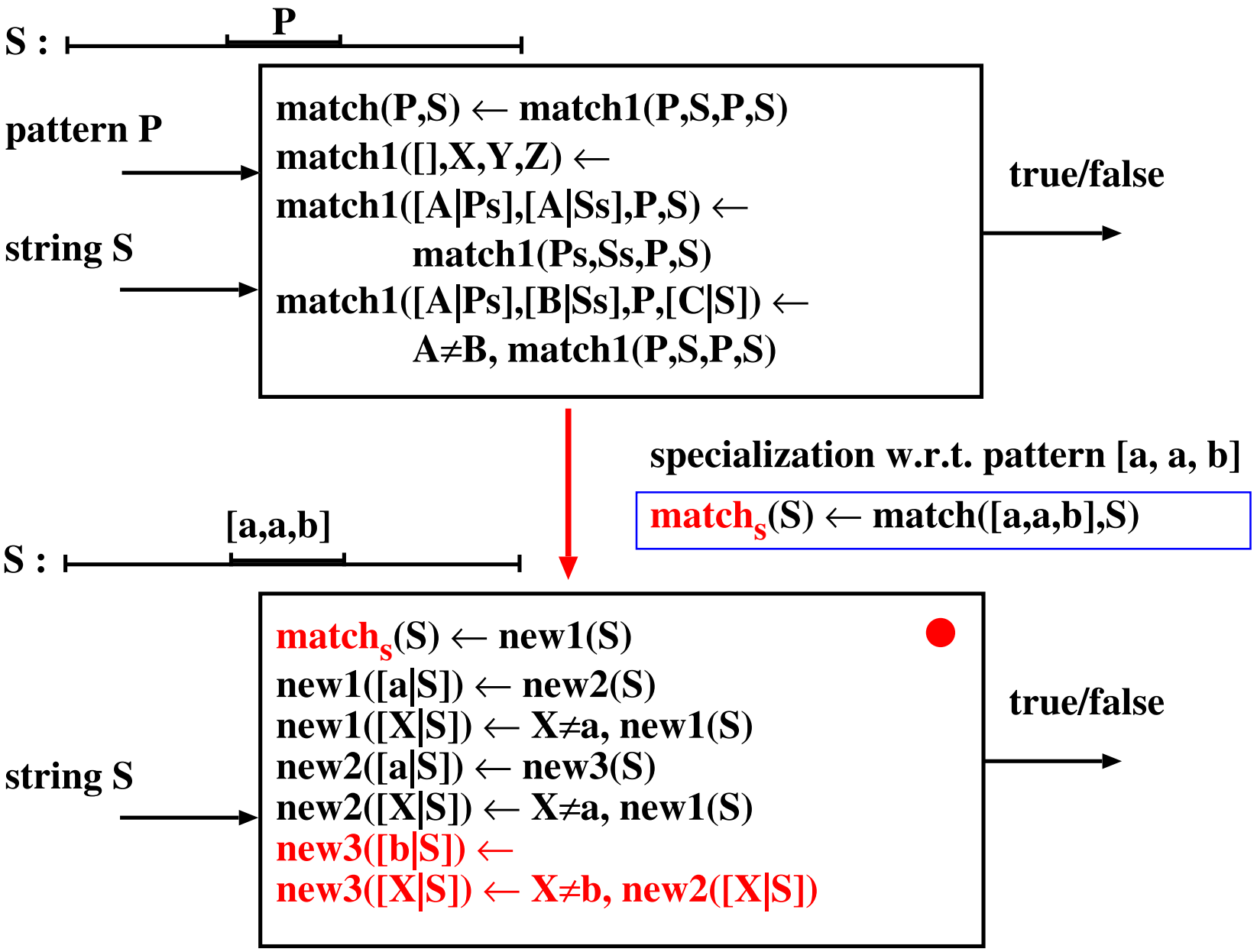
techniques: **Partial Evaluation**

Gallagher
Leuschel et al.
De Schreye et al.
Futamura et al.
Jones-Gomard-Sestoft
Lloyd-Shepherdson
Glück et al.

Program Transformation via Rules + Strategies

Burstall-Darlington
Tamaki-Sato

Specializing a Matcher (1)



[Fujita 87, Futamura 87, Glück-Klimov 93, Gallagher 93]

Specializing a Matcher (2)

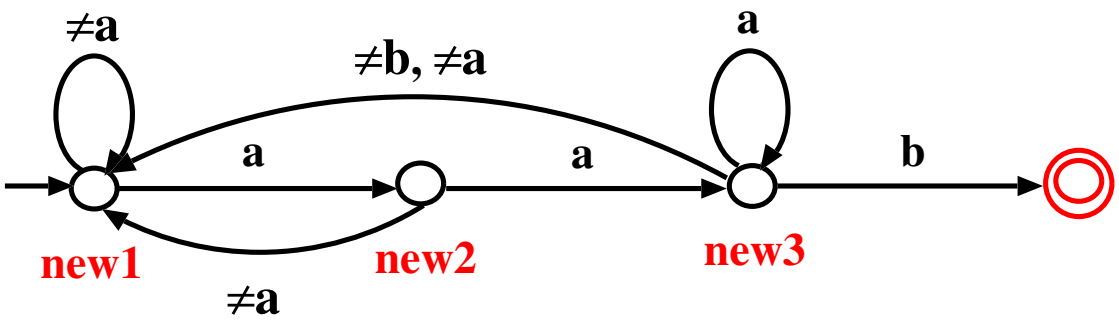
By unfolding we get :

string S →

$match_s(S) \leftarrow new1(S)$
$new1([a|S]) \leftarrow new2(S)$
$new1([X|S]) \leftarrow X \neq a, new1(S)$
$new2([a|S]) \leftarrow new3(S)$
$new2([X|S]) \leftarrow X \neq a, new1(S)$
$new3([b|S]) \leftarrow$
$new3([a|S]) \leftarrow new3(S)$
$new3([X|S]) \leftarrow X \neq b, X \neq a, new1(S)$

→ true/false

Finite automaton (Knuth-Morris-Pratt) :



- Clauses correspond to transitions and are mutually exclusive
- Predicates correspond to states

Plan

- Partial Evaluation as Program Transformation

- Limitations of Partial Evaluation

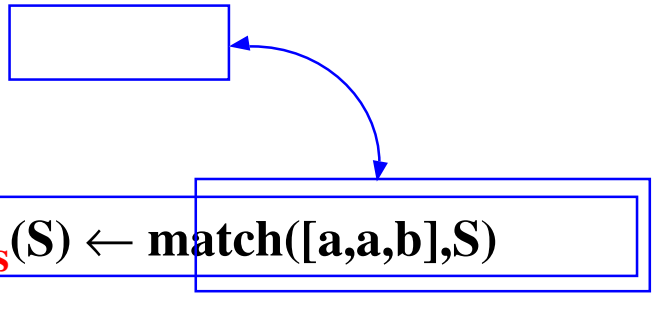
- Enhancing Partial Evaluation :

  - new rules for specialization
  - derivation of efficient matchers and parsers

Partial Deduction

We consider definite logic programs
with basic atoms: $X=t$ and $X\neq t$
Their semantics is the least Herbrand model M

Given a program P
an atomic, partially instantiated atom p(t(X))
(i.e., the static_input)



we introduce a clause $p_S(X) \leftarrow p(t(X))$ and

we look for a program P_S defining the predicate p_S

such that

$\forall$ ground term d (i.e., the dynamic_input)

$M(P) \models p(t(d))$ iff $M(P_S) \models p_S(d)$

Transformation via Rules + Strategies

initial program $P_0 \longrightarrow \ldots \longrightarrow P_n$ final program

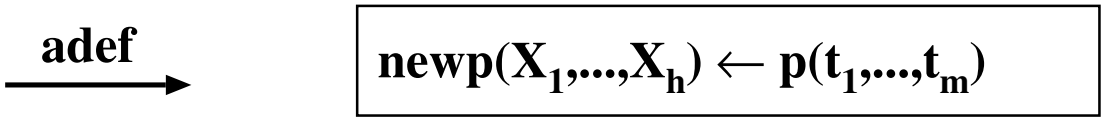
$$M(P_0) = \ldots = M(P_n)$$

- each rule transforms a program P_i into a new program P_{i+1}

- each rule preserves the least Herbrand model M

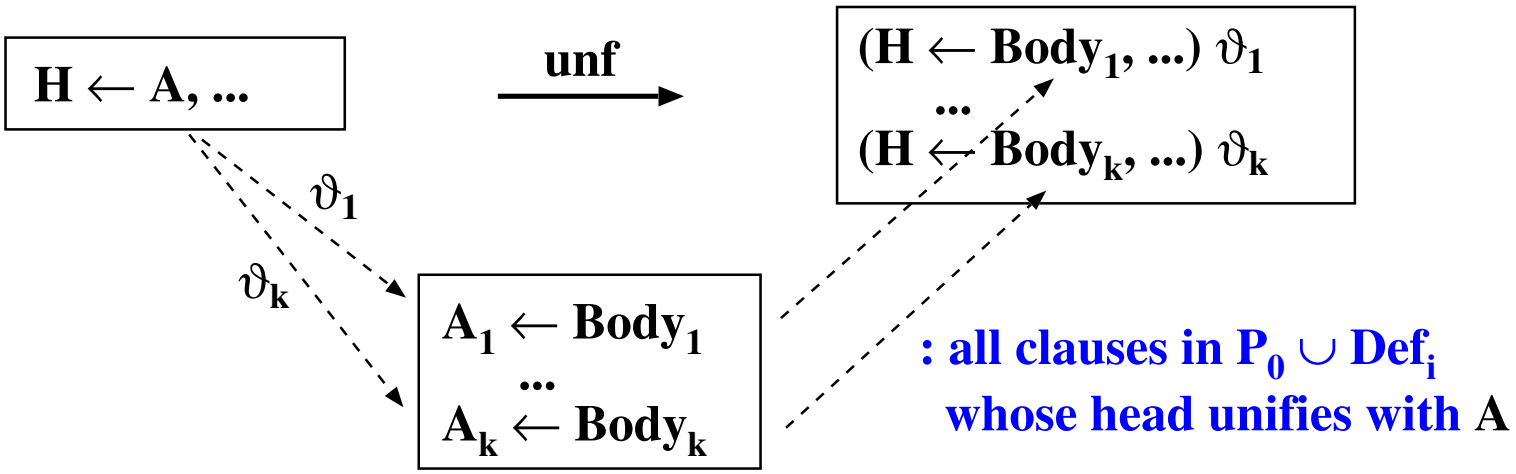
RULES for Partial Evaluation

Atomic Definition :

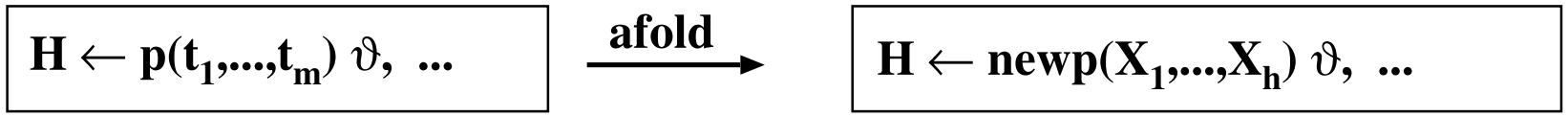


Def_i : all definitions introduced until P_i

Unfolding :



Atomic Folding :



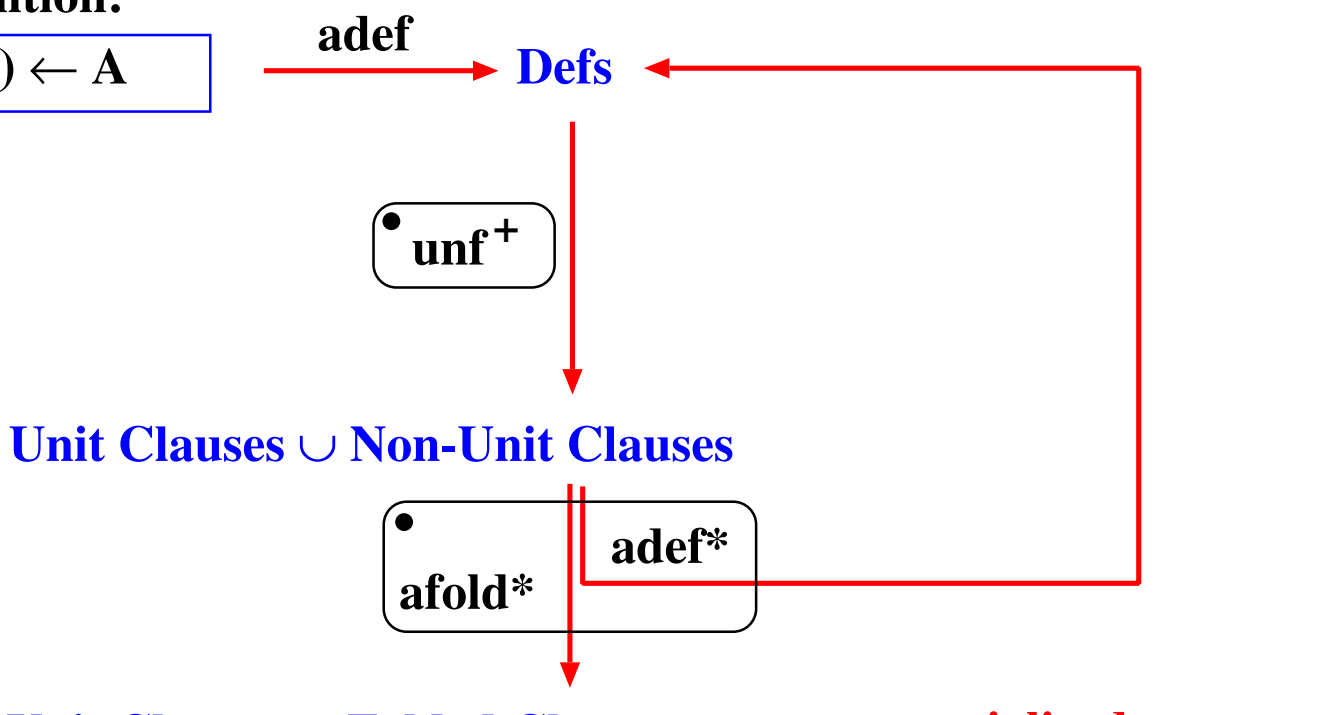
[Lloyd-Shepherson 91, Tamaki-Sato 84, Sahlin 91]

STRATEGY for Partial Evaluation

initial program P_0

first atomic definition:

$p_s(X_1,...,X_h) \leftarrow A$



- Halt when no new definitions are needed for folding
- Need for substrategies for unf+, afold*, adef*
- Unit clauses are clauses with empty body: H(...) ←

INITIAL PROGRAM:

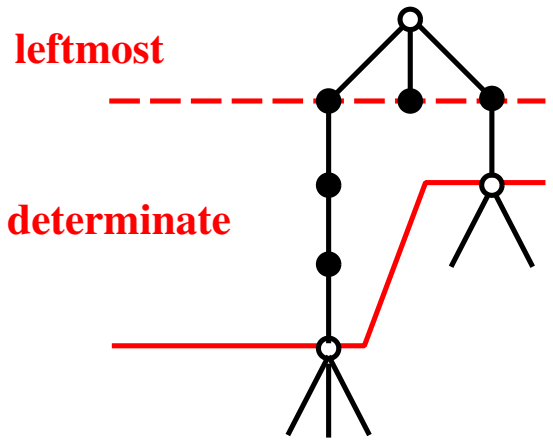
match(P,S) ← match1(P,S,P,S)
match1([],X,Y,Z) ←
match1([A|Ps],[A|Ss],P,S) ← match1(Ps,Ss,P,S)
match1([A|Ps],[B|Ss],P,[C|S]) ← A≠B, match1(P,S,P,S)

First atomic definition:

$match_s$(S) ← match([a,a,b],S)

Substrategy for unfolding:

- leftmost atom in the body and then
- determinate unfolding of atoms which are not inequalities
  (i.e., only one clause head unifies)



FIRST ITERATION:

First atomic definition:

1. $match_s(S) \leftarrow match([a,a,b],S)$

unf

2. $match_s(S) \leftarrow match1([a,a,b],S,[a,a,b],S)$

adef

more than one clause head unifies: we stop unfolding

afold

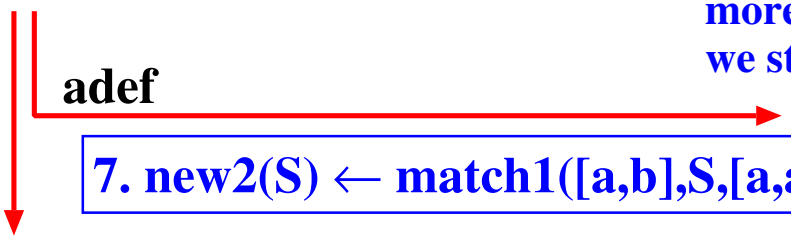
3. $new1(S) \leftarrow match1([a,a,b],S,[a,a,b],S)$

4. $match_s(S) \leftarrow new1(S)$

SECOND ITERATION:

3. new1(S) ← match1([a,a,b],S,[a,a,b],S)

5. new1([a|S]) ← match1([a,b],S,[a,a,b],[a|S])
6. new1([X|S]) ← X≠a, match1([a,a,b],S,[a,a,b],S)

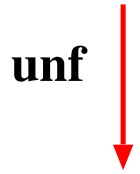


7. new2(S) ← match1([a,b],S,[a,a,b],[a|S])

8. new1([a|S]) ← new2(S)
9. new1([X|S]) ← X≠a, new1(S)

THIRD ITERATION:

7. new2(S) ← match1([a,b],S,[a,a,b],[a|S])



10. new2([a|S]) ← match1([b],S,[a,a,b],[a,a|S])
11. new2([X|S]) ← X≠a, match1([a,a,b],S,[a,a,b],S)

more than one clause head unifies: we stop unfolding

12. new3(S) ← match1([b],S,[a,a,b],[a,a|S])

13. new2([a|S]) ← new3(S)
14. new2([X|S]) ← X≠a, new1(S)

FOURTH ITERATION:

12. new3(S) ← match1([b],S,[a,a,b],[a,a|S])

unf

new3([b|S]) ←
match1([],S,[a,a,b],[a,a,b|S])

unf

15. new3([b|S]) ←

new3([X|S]) ← X≠b,
match1([a,a,b],[a,X|S],[a,a,b],[a,X|S])

afold

more than one clause head unifies: we stop unfolding

16. new3([X|S]) ← X≠b, new1([a,X|S])

By unfolding clause 16 we get:

string S →

4. $match_s$(S) ← new1(S) ●
8. new1([a|S]) ← new2(S)
9. new1([X|S]) ← X≠a, new1(S)
13. new2([a|S]) ← new3(S)
14. new2([X|S]) ← X≠a, new1(S)
15. new3([b|S]) ←
16* new3([X|S]) ← X≠b, new2([X|S])

→ **true/false**

Plan

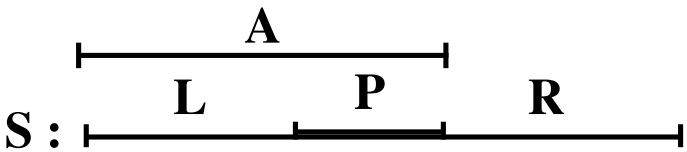
- Partial Evaluation as Program Transformation ✓

- Limitations of Partial Evaluation

- Enhancing Partial Evaluation :

  - new rules for specialization
  - derivation of efficient matchers and parsers

Limitations of Partial Evaluation (1)



pattern P

string S

```
match(P,S) ← append(A,R,S), append(L,P,A)
append([],L,L) ←
append([X|Xs],Ys,[X|Zs]) ← append(Xs,Ys,Zs)
```

true/false

specialization w.r.t. pattern [a, a, b]

$match_s(S) \leftarrow match([a,a,b],S)$

string S

IMPOSSIBLE to get
a deterministic finite automaton

true/false

Limitations of Partial Evaluation (2)

We get a nondeterministic program :

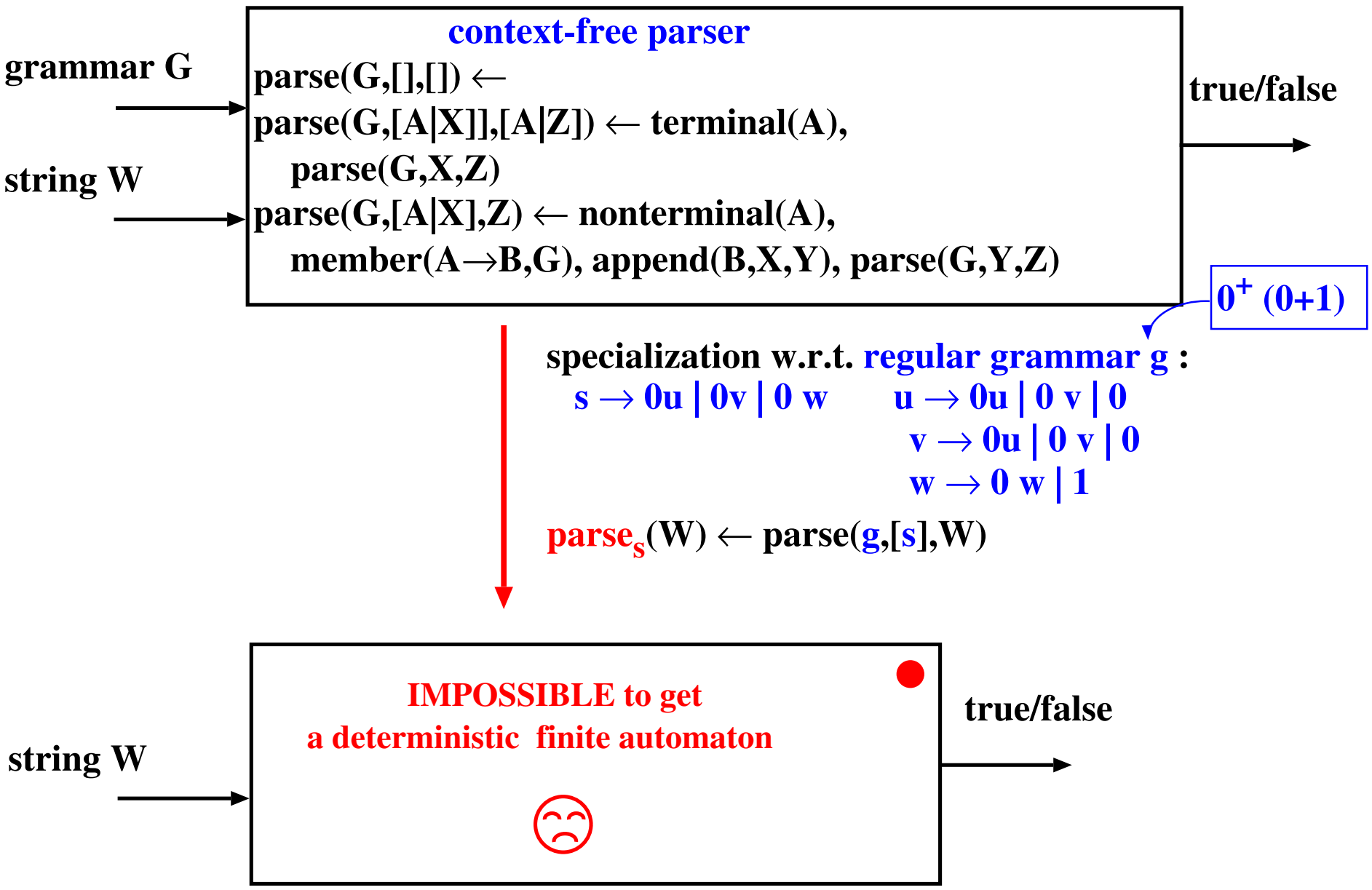
$match_s([a,a,b|S]) \leftarrow$

$match_s([A|T]) \leftarrow new1(X,Y), append(Y,Z,T)$
$new1([],[a,a,b]) \leftarrow$
$new1([H|T],[H|U]) \leftarrow new1(T,U)$
$append([],L,L) \leftarrow$
$append([X|Xs],Ys,[X|Zs]) \leftarrow append(Xs,Ys,Zs)$

☹ **Look-ahead of 3 symbols, even if we allow non-atomic definitions**
(Conjunctive Partial Deduction by Leuschel et al.)

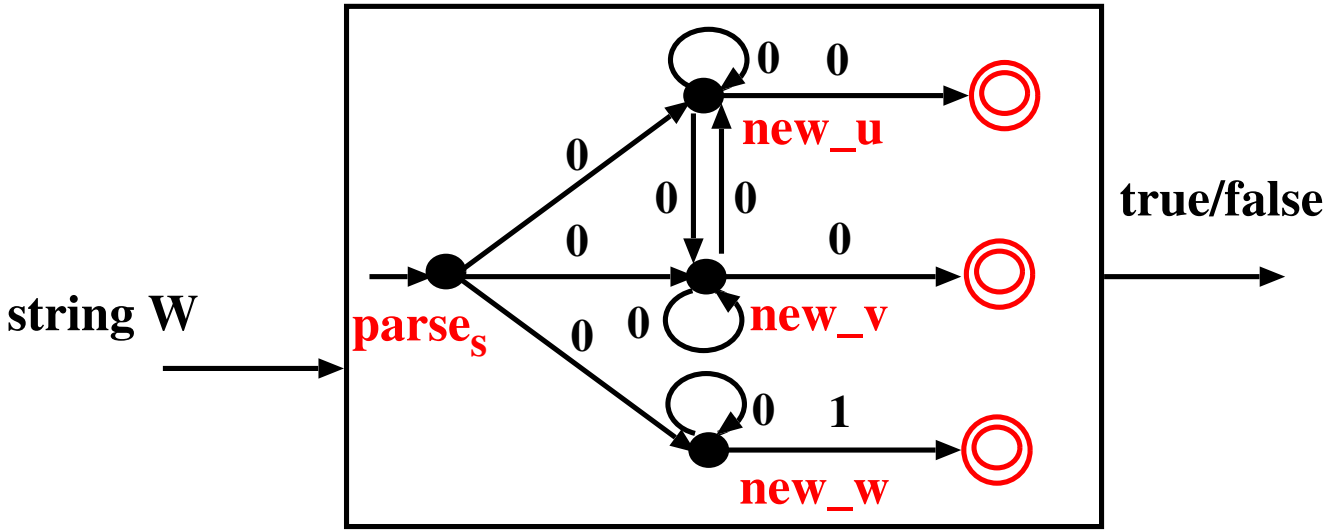
☹ **Not mutually exclusive clauses**

Limitations of Partial Evaluation (3)

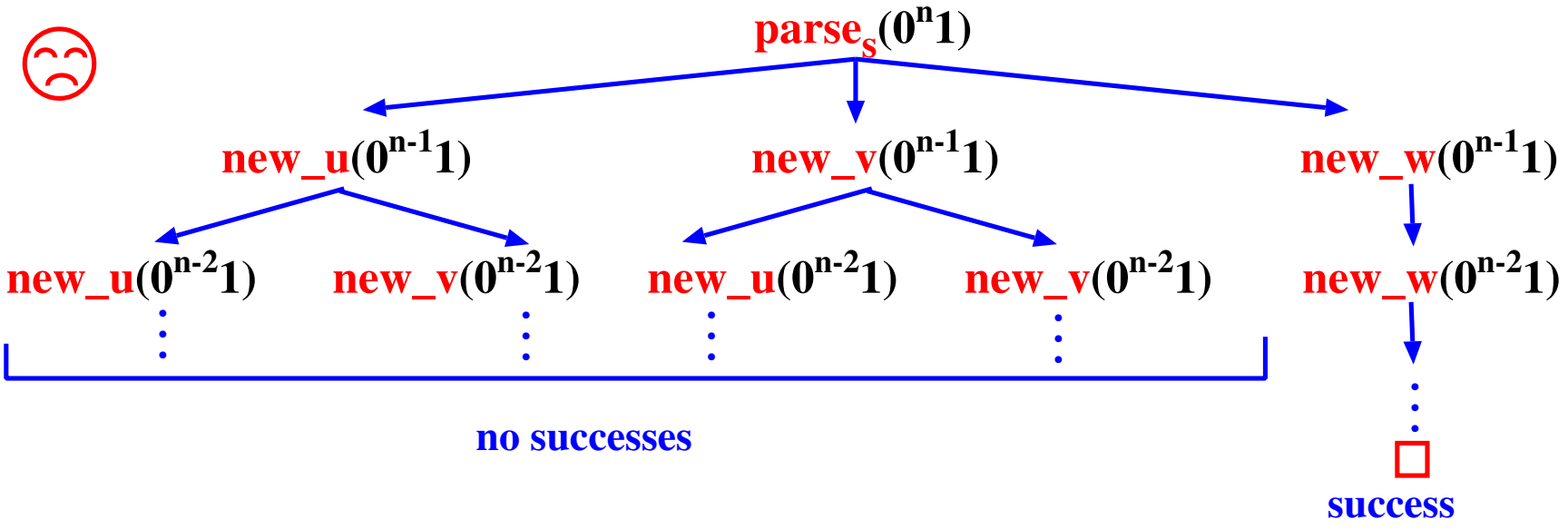


Limitations of Partial Evaluation (4)

We get a **nondeterministic finite automaton** :

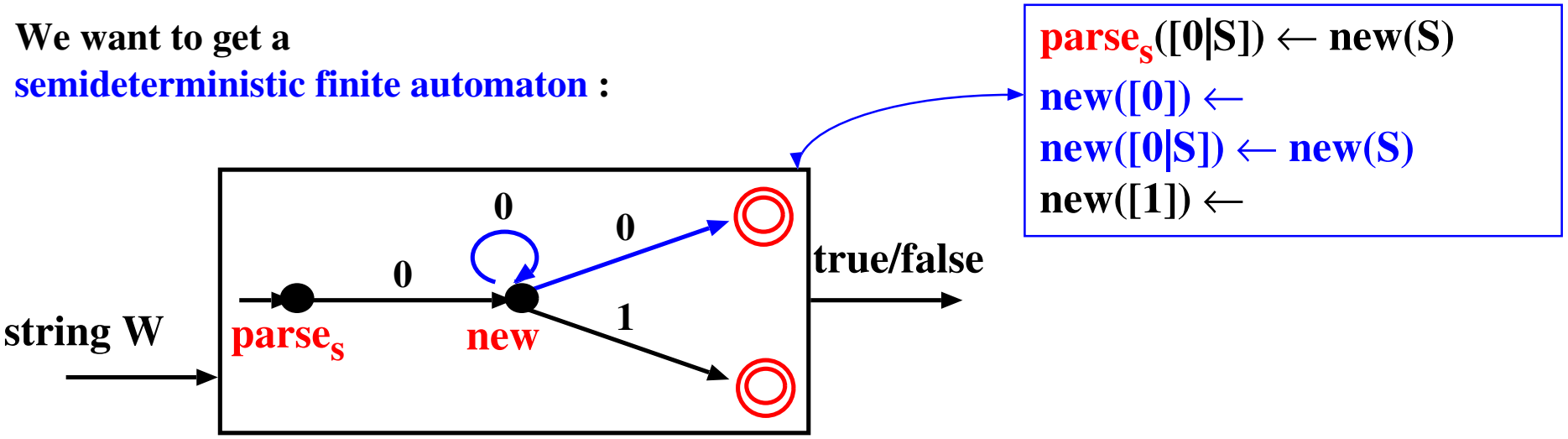


EXPONENTIAL SEARCH :

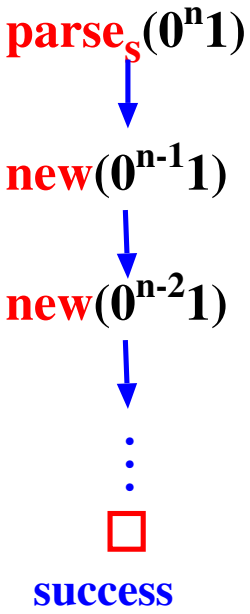


Desiderata for Specialization

We want to get a
semideterministic finite automaton :



LINEAR SEARCH :



Plan

- Partial Evaluation as Program Transformation ✓

- Limitations of Partial Evaluation ✓

- Enhancing Partial Evaluation :

  - new rules for specialization
  - derivation of efficient matchers and parsers

Basic Ideas (1)

- increase of determinism
- generation of mutually exclusive clauses :

goal : A clauses : $H_1 \leftarrow d_1 , G_1$; $H_2 \leftarrow d_2 , G_2$

if both $A=H_1 , d_1$ and $A=H_2 , d_2$ are satisfiable

then split :

$H_1 \leftarrow d_1 , G_1$

$H_2 \leftarrow d_1 , d_2 , G_2$

$H_2 \leftarrow \neg d_1 , d_2 , G_2$

two mutually exclusive packets of clauses

$H_1 \leftarrow d_1 , G_1$

$H_2 \leftarrow d_1 , d_2 , G_2$

folding → $\gamma 1$

$H_2 \leftarrow \neg d_1 , d_2 , G_2$ → $\gamma 2$

Basic Ideas (2)

- **mutual exclusive clauses imply determinism**

  if variables are sufficiently instantiated by d_1 and d_2 :

  mode analysis needed.

NEW RULES

Operational Semantics and Modes

NEW RULES for Specialization (1)

Definition :

$$\xrightarrow{\text{def}}$$

$$\begin{array}{l} newp(X_1,...,X_h) \leftarrow Body_1 \\ \quad ... \\ newp(X_1,...,X_h) \leftarrow Body_n \end{array}$$

$X_1,...,X_h$ are some of the variables in $Body_1,...,Body_n$

Unfolding : (as before)

Folding :

$$\begin{array}{l} H \leftarrow Body_1\,\vartheta,\ ... \\ \quad ... \\ H \leftarrow Body_n\,\vartheta,\ ... \end{array}$$

$$\xrightarrow{\text{fold}}$$

$$H \leftarrow newp(X_1,...,X_h)\,\vartheta,\ ...$$

NEW RULES (2)

Subsumption :

$H \leftarrow Body$

$(H \leftarrow Body, G)\ \vartheta$

$\longrightarrow$ $H \leftarrow Body$

Head generalization :

C: $H\{X/t\} \leftarrow Body$ $\longrightarrow$ $H \leftarrow X=t, Body$

if X not in C

Case split :

$H(X) \leftarrow Body$ $\longrightarrow$

$H(t) \leftarrow Body\{X/t\}$

$H(X) \leftarrow X \neq t, Body$

Equation elimination :

$H \leftarrow G_1, t_1=t_2, G_2$ $\longrightarrow$

if ϑ is the mgu of t_1 and t_2
then $(H \leftarrow G_1, G_2)\ \vartheta$
else remove

NEW RULES (3)

Disequation replacement :

1. $H \leftarrow G_1, t_1 \neq t_2, G_2 \longrightarrow H \leftarrow G_1, G_2$

   if t_1 and t_2 are not unifiable

2. $H \leftarrow G_1, f(t_1,...,t_m) \neq g(u_1,...,u_m), G_2 \longrightarrow$

   $H \leftarrow G_1, t_1 \neq u_1, G_2$

   ...

   $H \leftarrow G_1, t_m \neq u_m, G_2$

3. $H \leftarrow G_1, X \neq X, G_2$ is removed

4. $H \leftarrow G_1, t \neq X, G_2 \longrightarrow H \leftarrow G_1, X \neq t, G_2$

5. $H \leftarrow G_1, X \neq t_1, G_2, X \neq t_2, G_3 \longrightarrow H \leftarrow G_1, X \neq t_1, G_2, G_3$

   if $\exists$ a remaning ρ of the local variables of t_1 such that $t_1\rho = t_2$

NEW RULES : Correctness

Theorem.

If all definition clauses used for folding, are unfolded,

then the new rules preserve the Least Herbrand Model.

Operational Semantics : $\mapsto$

a program P

basic atoms:

(1) $t_1 = t_2$, G $\mapsto$ Gϑ iff $\vartheta = mgu(t_1,t_2)$

(2) $t_1 \neq t_2$, G $\mapsto$ G iff t_1 and t_2 are not unifiable

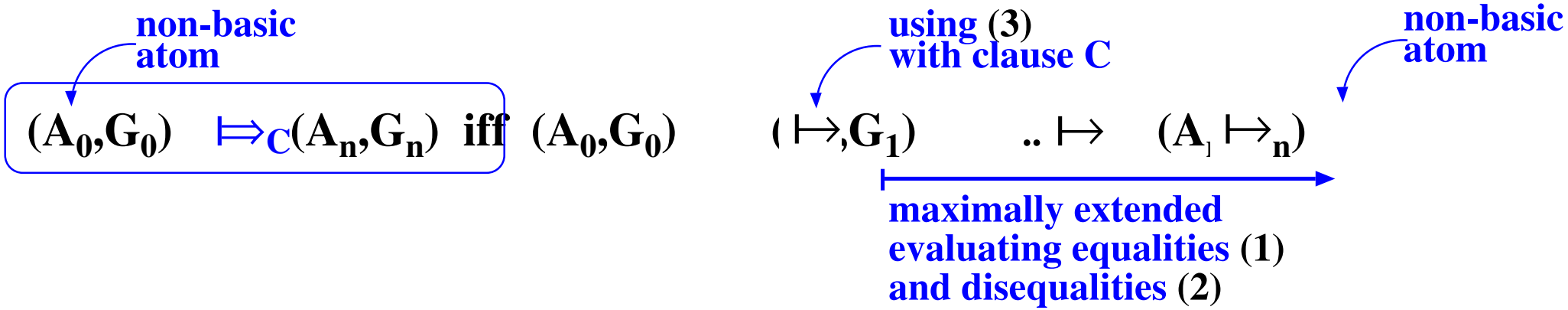
non-basic atoms:

(3) A, G $\mapsto$ (body(C), G) ϑ iff C is a clause of P and
$\vartheta = mgu(A,head(C))$

a goal G

successful derivation: G tr $\mapsto^*$

Determinism

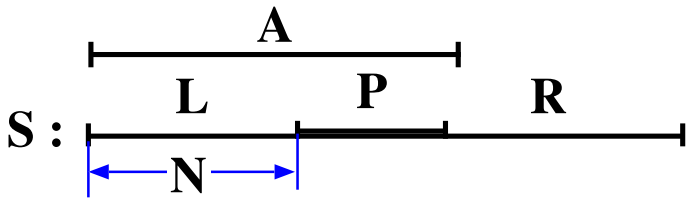


A program P is deterministic for an atom A iff for each A $\mapsto_{C1} \ldots \mapsto_{Ck} G$ there exists at most one clause C in P s.t. $G \mapsto_C G1$ for some goal G1

Thus, if a program P is deterministic for an atom A and A $\mapsto^*$ true then there exists at most one successful derivation for A.

Semideterminism

Many programs have more than one successful derivation.




```
match_pos(P,S,N) ← append(L,P,A), append(A,R,S), length(L,N)
length([],0) ←
length([X|Xs],s(N)) ← length(Xs,N)
append([],L,L) ←
append([X|Xs],Ys,[X|Zs]) ← append(Xs,Ys,Zs)
```

A program P is **semideterministic** for an atom A iff for each A $\mapsto_{C1}$.. $\mapsto_{Ck}$G there exists **at most one clause** C in P s.t. G $\mapsto_{C}$ G1 for some goal G1 $\neq$ **true**

Example.

```
parse_s([0|S]) ← new(S)
new([0]) ←
new([0|S]) ← new(S)
new([1]) ←
```

is semideterministic for $parse_s$(S)

Modes (1)

- a mode for a predicate p : p(..., +, ..., ?, ...)

  ground no information

  an input term is a term with +
  an input variable is a variable in a term with +

- an atom p(...) satisfies a mode M
  iff every term in p(...) with + is ground

- a program P satisfies a mode M

  iff for each atom A_0 which satisfies M and

  for each non-basic atom A and goal G s.t. A_0 $(A,G) \mapsto^*$

  A satisfies M

Modes (2)

- a program P is semideterministic w.r.t. a mode M iff
  P is semideterministic for each atom which satisfies M

- Theorem (sufficient conditions for semideterminism)
  If P satisfies M,
  P is linear,
  P is safe, and
  the non-unit clauses of P are pairwise
  mutually exclusive
  then P is semideterministic w.r.t. a mode M

- **P is linear : at most one non-basic atom in the body**

$$H(...) \leftarrow X_1=t_1, ..., X_r=t_r, ..., p(...)$$

- **P is safe : each variable X in a disequation $t_1 \neq t_2$ of a clause C**

  either is an input variable of hd(C)

  or is a local variable of $t_1 \neq t_2$ of C

- **Given a mode, say p(+,?),**
  the two clauses

  leftmost no-basic atoms

  C_1: $p(t_1,u_1) \leftarrow X_{11} \neq v_{11}, ..., p_1(...), ...$

  C_2: $p(t_2,u_2) \leftarrow X_{12} \neq v_{12}, ..., p_2(...), ...$

  are mutually exclusive clauses iff either t_1 and t_2 are not unifiable

  or t_1 and t_2 are unifiable via the mgu ϑ

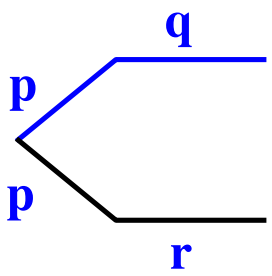
  and $X_{11} \neq v_{11}, ... \ \vartheta$ are not satisfiable

  $X_{12} \neq v_{12}, ...$

MUTUALLY EXCLUSIVE CLAUSES

1. p([t|Y]) ← q(Y)
2. p([X|Y]) ← r(Y)

: not mutually exclusive clauses w.r.t. mode p(+)



case split w.r.t. X/t

1. p([t|Y]) ← q(Y)
2.1 p([t|Y]) ← r(Y)

packet 1

2.2 p([X|Y]) ← X=t, r(Y)

packet 2

new1(Y) ← q(Y)
new1(Y) ← r(Y)
new2(Y) ← r(Y)

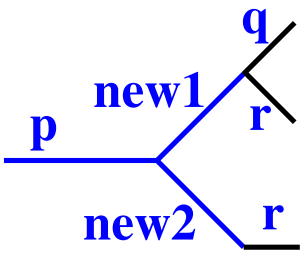
def*

fold*

p([t|Y]) ← new1(Y)

p([X|Y]) ← X=t, new2(Y)

: mutually exclusive clauses w.r.t. mode p(+)

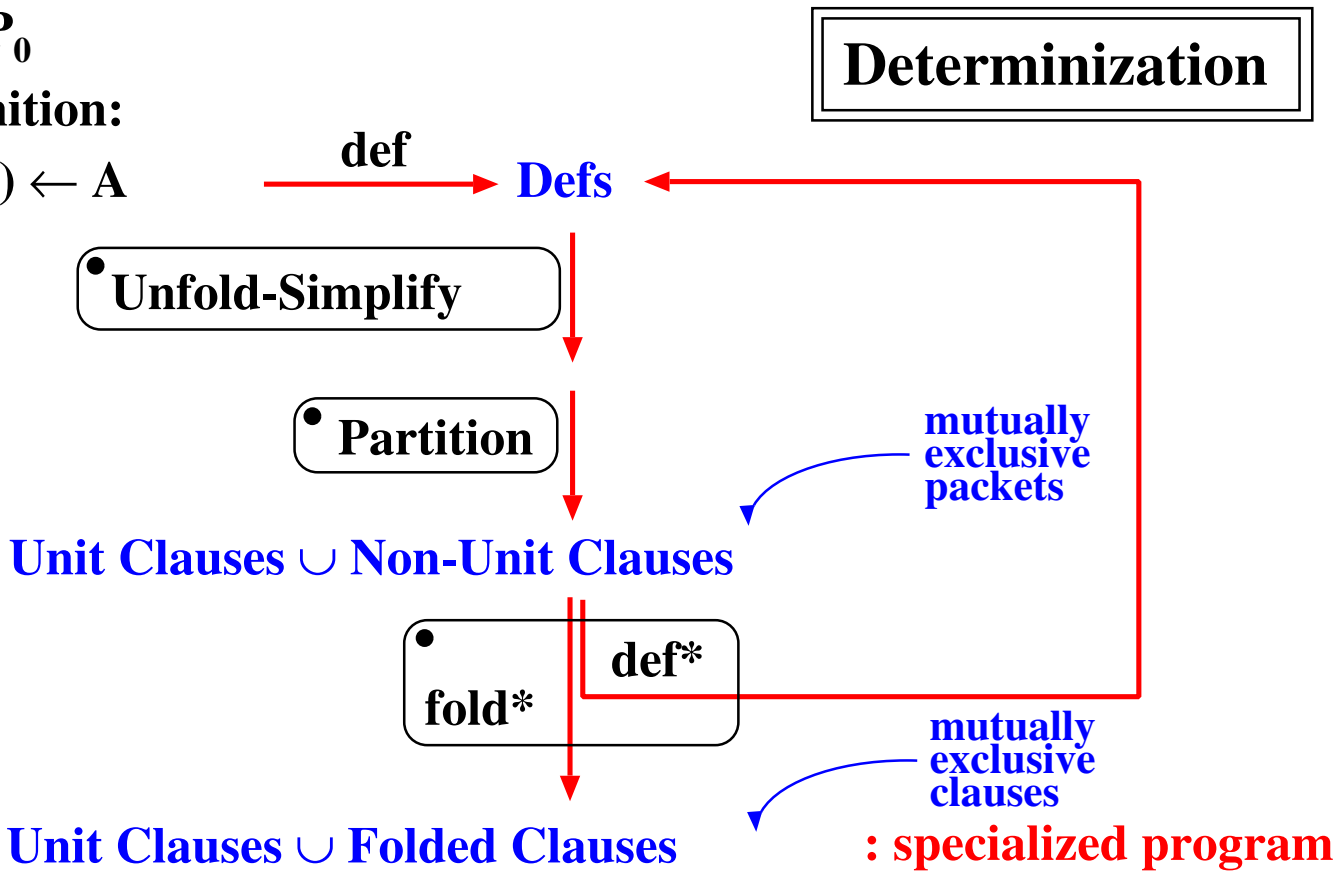


NEW STRATEGY for Specialization

initial program P_0

first atomic definition:

$p_s(X_1,...,X_h) \leftarrow A$



- Halt when no new definitions are needed for folding
- Need for substrategies for

This new strategy improves over :

- conjunctive partial deduction (Leuschel et al. 96)

- elimination of unnecessary variables
(argument filtering, Leuschel-Sørensen 96)

- elimination of redundant structures
(renaming, Gallagher et al. 90, Benkerimi-Hill 93)

Related work:
- deforestation (Wadler 80)
- tupling (Pettorossi 77, Chin 90]
- supercompilation (Turchin86, Glück-Sørensen 94)

Substrategy for Unfold-Simplify

- *unfold* the leftmost non-basic atom,
- *simplify* the derived clauses, and
- *as long as possible*
    1. *unfold* the leftmost non-basic atom whose input arguments are *not* instantiated when unfolding, and
    2. *simplify* clauses.

simplify a clause =
as long as possible do
    2.1 *eliminate* equations,
    2.2 *eliminate* disequations and *move* disequations to the left, and
    2.3 *apply* subsumptions.

i.e., compute the "normal form"

Substrategy for **Partition**

- *as long as possible*

  1. take any two non mutually exclusive clauses and apply *case split*, and
  2. *simplify* clauses

  : we get mutually exclusive packets of clauses.

- apply *head generalization* so to replace the non-input arguments in the heads of the clauses in the same packet by their most specific common generalization, and
- *move* all disequations to the leftmost positions.

Substrategy for Define-Fold

- use already introduced definitions, if possible,
  otherwise introduce a set of new clauses which allows folding.

INITIAL PROGRAM:

match_pos(P,S,N) ← append(A,R,S), append(L,P,A), length(L,N)

length([],0) ←

length([X|Xs],s(N)) ← length(Xs,N)

append([],L,L) ←

append([X|Xs],Ys,[X|Zs]) ← append(Xs,Ys,Zs)

First atomic definition:

$match_pos_s$(S,N) ← match_pos([a,a,b],S,N)

Mode M: $match_pos_s(+,?)$ match_pos(+,+,?) append(?,?,+) length(+,?)

FIRST ITERATION:

First atomic definition:

1. $match_pos_s(S,N) \leftarrow match_pos([a,a,b],S,N)$

unf

these input variables get instantiated when unfolding

2. $match_pos_s(S) \leftarrow append(A,R,S), append(L,[a,a,b],A),$
   $length(L,N)$

def

3. $new1(S,N) \leftarrow append(A,R,S), append(L,[a,a,b],A),$
   $length(L,N)$

fold

4. $match_s(S,N) \leftarrow new1(S,N)$

SECOND ITERATION:

3. . new1(S,N) ← append(A,R,S), append(L,[a,a,b],A),
length(L,N)

unf+

these input variables get instantiated when unfolding

5. new1([a|S],0) ← append([a,b],R,S)

6. new1([C|S],s(N)) ← append(A,R,S), append(L,[a,a,b],A),
length(L,N)

case split &
head general.

5.1 new1([a|S],M) ← M=0, append([a,b],R,S)

6.1 new1([a|S],M) ← M=s(N), append(A,R,S), append(L,[a,a,b],A),
length(L,N)

6.2 new1([C|S],s(N)) ← C≠a, append(A,R,S), append(L,[a,a,b],A),
length(L,N)

def

7. new2(S,M) ← M=0, append([a,b],R,S)

8. new2(S,M) ← M=s(N), append(A,R,S), append(L,[a,a,b],A),
length(L,N)

fold

8. new1([a|S],M) ← new2(S,M)

9. new1([C|S],s(N)) ← C≠a, new1(S,N)

FINAL PROGRAM : semideterministic w.r.t. mode M

Nondeterminism is due only to multiple solutions

4. $match_s$(S,N) ← new1(S,N)

8. new1([a|S],M) ← new2(S,M)
9. new1([C|S],s(N)) ← C≠a, new1(S,N)

17. new2([a|S],N) ← new3(S,N)
18. new2([C|S], s(s(N))) ← C≠a, new1(S,N)

25. new3([b|S],N) ← new4(S,N)
26. new3([a|S], s(N)) ← new3(S,N)
27. new3([C|S], s(s(s(N)))) ← C≠b, C≠a, new1(S,N)

28. new4(S,0) ←
31. new4([a|S], s(s(s(N)))) ← new2(S,N)
32. new4([C|S], s(s(s(s(N))))) ← C≠a, new1(S,N)

Multiple occurrences of the patterns are found :

for P = [a, a, b] and S = [a, a, b, a, a, a, b, a, a, b, b] we get 1, 5, 8.

FINAL PROGRAM : semideterministic w.r.t. mode M

with cuts, instead of ≠

4. $match_s$(S,N) ← new1(S,N)

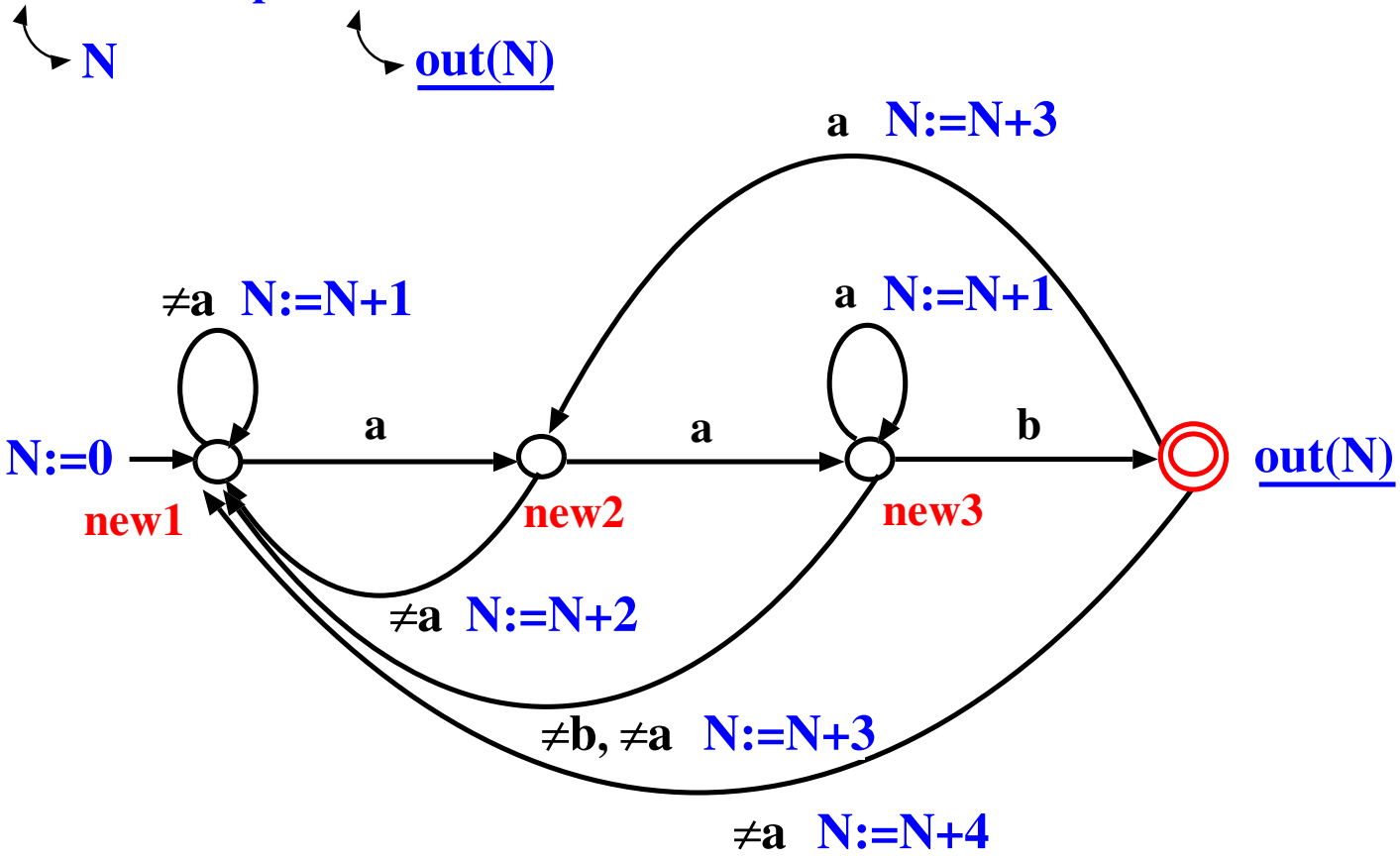
8. new1([a|S],M) ← !, new2(S,M)
9. new1([C|S],s(N)) ← new1(S,N)

17. new2([a|S],N) ← !, new3(S,N)
18. new2([C|S], s(s(N))) ← new1(S,N)

25. new3([b|S],N) ← !, new4(S,N)
26. new3([a|S], s(N)) ← !, new3(S,N)
27. new3([C|S], s(s(s(N)))) ← new1(S,N)

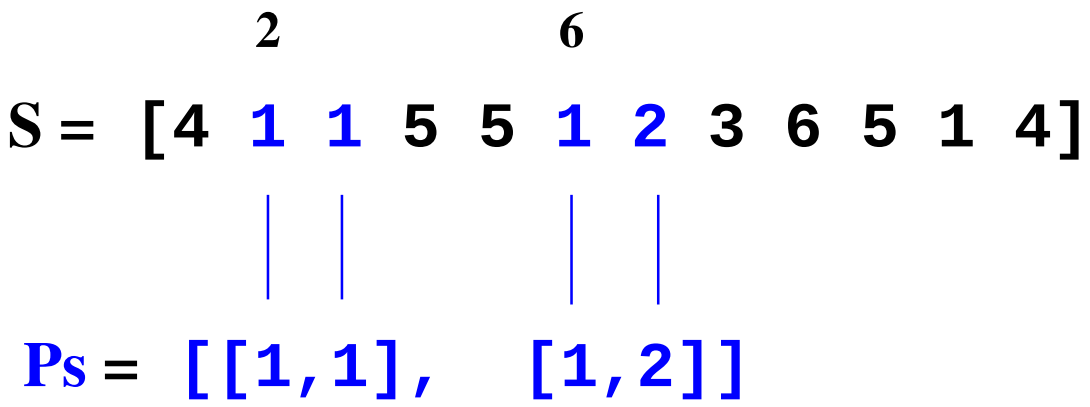
28. new4(S,0) ←
31. new4([a|S], s(s(s(N)))) ← !, new2(S,N)
32. new4([C|S], s(s(s(s(N))))) ← new1(S,N)

The corresponding deterministic finite automaton
with counters and output is :



Multimatch

mmatch([P|Ps], S, N) ← match_pos(P, S, N)
mmatch([P|Ps], S, N) ← mmatch(Ps, S, N)



N = 2, 6

Regular Expressions

in_language(E,S) ← string(S), accepts(E,S)
string([ ]) ←
string([a|S]) ← string(S)
string([b|S]) ← string(S)
accepts(E,[E]) ← symbol(E)
accepts(E,[E]) ← symbol(E)
accepts(E_1E_2,S) ← accepts(S_1,S_2,S), accepts(E_1,S_1), accepts(E_2,S_2)
accepts(E_1+E_2,S) ← accepts(E_1,S_1)
accepts(E_1+E_2,S) ← accepts(E_2,S_2)
accepts(E*,[ ]) ←
accepts(E*,S) ← ne_accepts(S_1,S_2,S), accepts(E,S_1), accepts(E*,S_2)
symbol(a) ←
symbol(b) ←
ne_append([A],Y,[A|Y]) ←
ne_append([A|X],Y,[A|Z]) ← ne_append(X,Y,Z)

in_language(E,S) succeeds iff S is a word in the regular expression E.

Specialization w.r.t. E = (aa*(b+bb))*

E = a*(b+bb+bbb)

Matching Regular Expressions

re_match(E,S) ← append(A,R,S), append(L,P,A), accepts(E,P)

re_match(E,S) succeeds iff a word in E is a substring of S.

Specialization w.r.t. E = aa*b

E = a*(b+bb)

Context-free parsing

```
string_parse(G,A,W) ← string(W), parse(G,A,W)
string([ ]) ←
string([0|W]) ← string(W)
string([1|W]) ← string(W)
parse(G,[],[]) ←
parse(G,[A|X]],[A|Z]) ← terminal(A), parse(G,X,Z)
parse(G,[A|X],Z) ← nonterminal(A),
                   member(A→B,G), append(B,X,Y), parse(G,Y,Z)
member(A,[A|X]) ←
member(A,[B|X]) ← member(A,X)
```

string_parse(G,[s],W) succeeds iff
the string W is generated by the grammar G with start symbol s from the list [s]

Specialization w.r.t. grammar

$g = s \rightarrow 0u \mid 0v \mid 0w \quad u \rightarrow 0 \mid 0u \mid 0v \quad v \rightarrow 0 \mid 0v \mid 0u \quad w \rightarrow 1 \mid 0w$

$g_1 = s \rightarrow 0u \mid 1v$

$u \rightarrow 0 \mid 0v \mid 0w \quad v \rightarrow 1 \mid 0v \mid 1u \quad w \rightarrow 1 \mid 1w$

Specialization time

	ECCE (PD) ms	MAP (Det) ms
match([aab],S)	360	70
match([aaaaaaaaab],S)	420	480
match_pos([aab],S,N)	540	100
match_pos([aaaaaaaaab],S,N)	650	500
mmatch([[aaa],[aab]],S,N)	1150	280
mmatch([[aa],[aaa],[aab]],S,N)	1740	220
in_language((aa*(b+bb))*,S)	6260	420
in_language(a*(b+bb+bbb),S)	3460	230
re_match(aa*b,S)	970	210
re_match(a*(b+bb),S)	1970	300
string_parse(g,[s],W)	23400	1620
string_parse(g_1,[s],W)	31200	2000

Speedups

	Speedup (PD)	Speedup (Conj. PD)	Speedup (Det)	Speedup (Det & Cut)
match([aab],S)	3.1	5.8 x 10^3	3.0 x 10^3	6.8 x 10^3
match([aaaaaaaaab],S)	3.3	6.9 x 10^3	5.8 x 10^3	12.4 x 10^3
match_pos([aab],S,N)	1.6	3.6 x 10^3	1.8 x 10^3	4.0 x 10^3
match_pos([aaaaaaaaab],S,N)	2.1	5.3 x 10^3	2.9 x 10^3	8.1 x 10^3
mmatch([[aaa],[aab]],S,N)	1.7	4.5 x 10^3	3.5 x 10^3	6.2 x 10^3
mmatch([[aa],[aaa],[aab]],S,N)	1.6	2.5 x 10^3	3.9 x 10^3	5.4 x 10^3
in_language((aa*(b+bb))*,S)	29.8	6.2 x 10^3	2.3 x 10^5	3.9 x 10^5
in_language(a*(b+bb+bbb),S)	1.3 x 10^4	3.3 x 10^3	4.6 x 10^4	5.7 x 10^4
re_match(aa*b,S)	5.7 x 10^2	2.7 x 10^3	1.5 x 10^6	3.0 x 10^6
re_match(a*(b+bb),S)	2.1 x 10^2	3.4 x 10^3	2.5 x 10^5	4.1 x 10^5
string_parse(g,[s],W)	1.5	1.5	87.1	87.1
string_parse(g_1,[s],W)	1.1	1.1	61.3	61.3

References

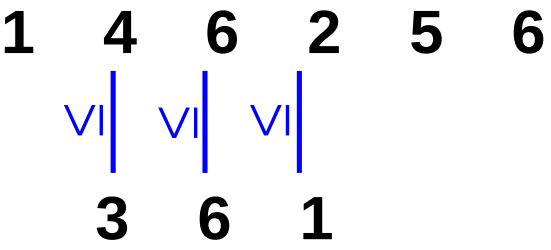
MAP Transformation System available at :

`http://www.iasi.rm.cnr.it/~proietti`

... and Future Work

- same classes of algorithms with generic constraints,

instead of = and ≠ only. For instance, ≤



- other classes of algorithms, e.g., scheduling algorithms

More slides on specialization (1)

Program Specialization

Given a locally stratified CLP(*D*) program P and

a constraint atom $c(X) \wedge p(X)$

Perfect *D*-model : M(P)

Derive an efficient program P_{sp} defining p_{sp} s.t.

$\forall$ ground term t

$M(P) \models c(t) \wedge p(t)$ iff $M(P_{sp}) \models c(t) \wedge p_{sp}(t)$

---

1. Introduce δ : $p_{sp}(X_1, ..., X_h) \leftarrow c(X) \wedge p(X)$

2. Apply rules according to a strategy: $P \cup \{\delta\} \xrightarrow{\text{rules}}{}^{*} P_{sp}$

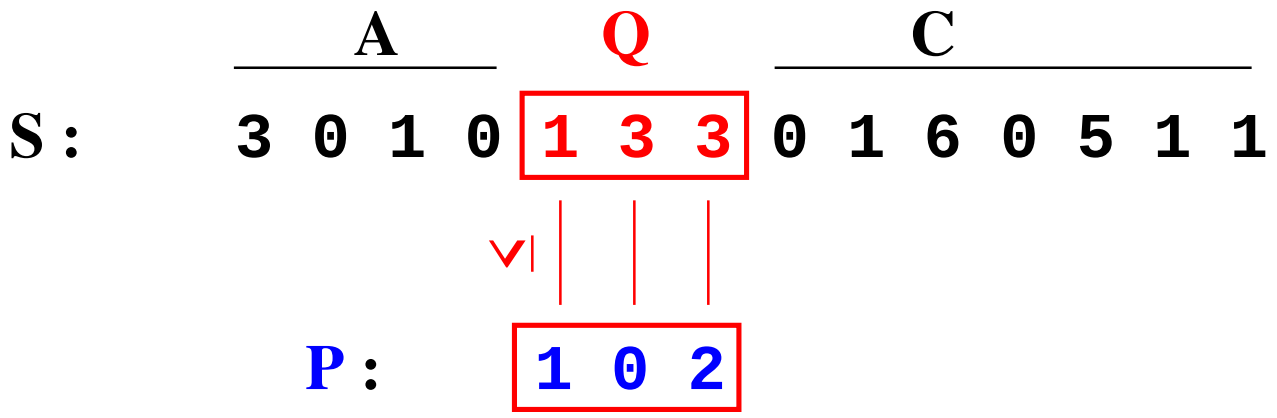
Lopstr 2000, AGP 2002

Leq_Match 1

1. leq_match(P,S) ← append(B,C,S) ∧ append(A,Q,B) ∧ leq(P,Q)

2. append([ ], Ys, Ys) ←
3. append([X|Xs], Ys, [X|Zs]) ← append(Xs, Ys, Zs)

4. leq([ ], [ ]) ←
5. leq([X|Xs], [Y|Ys]) ← X ≤ Y ∧ leq(Xs,Ys)



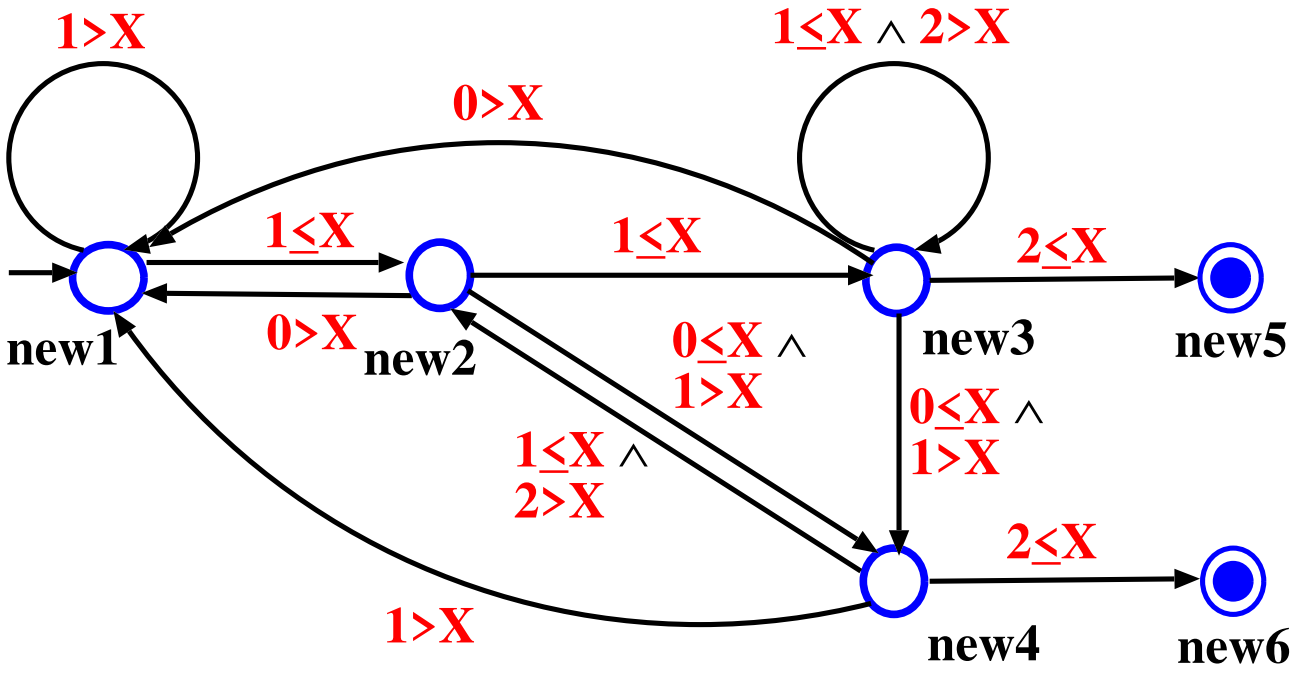
6. leq_match_{sp}(S) ← leq_match([1,0,2],S)

Leq_Match 2

by specializing w.r.t. the goal leq_match([1,0,2], S)
we get the deterministic program:

9. $leq_match_{sp}(S) \leftarrow new1(S)$
16. $new1([X|Xs]) \leftarrow 1 \leq X \wedge new2(Xs)$
17. $new1([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$
$new2([X|Xs]) \leftarrow 1 \leq X \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new2([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new3([X|Xs]) \leftarrow 2 \leq X \wedge new5(Xs)$
$new3([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new3(Xs)$
$new3([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new3([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new4([X|Xs]) \leftarrow 2 \leq X \wedge new6(Xs)$
$new4([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new2(Xs)$
$new4([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$
$new5(Xs) \leftarrow$
$new6(Xs) \leftarrow$

Leq_Match 3



efficiency= deterministic automaton

Basic Idea

- determinism = mutually exclusive clauses

clauses : $H_1 \leftarrow c_1 \wedge G_1$, $H_2 \leftarrow c_2 \wedge G_2$

goal : A

if both $c_1 \wedge A = H_1$, $c_2 \wedge A = H_2$ are satisfiable

then split :

$H_1 \leftarrow c_1 \wedge G_1$

$H_2 \leftarrow c_1 \wedge c_2 \wedge G_2$

$H_2 \leftarrow \neg c_1 \wedge c_2 \wedge G_2$

$\Rightarrow$

mutually exclusive packets

$H_1 \leftarrow c_1 \wedge G_1$

$H_2 \leftarrow c_1 \wedge c_2 \wedge G_2$

$H_2 \leftarrow \neg c_1 \wedge c_2 \wedge G_2$

multiple folding

$\gamma 1$

$\gamma 2$

Transformation Rules

R1. Definition
R2. Unfolding
R3. Folding
R4. Clause Removal
R5. Constraint Replacement
R6-R7. Clause Fusion and Clause Splitting

extension of rules of :
- **Bensaou-Guessarian**
- **Etalle-Gabbrielli**
- **Gergatsoulis-Katzouraki**
- **Tamaki-Sato**

R1. Definition

Introduce the clauses

$$\delta_1 : \quad newp(X_1, ..., X_h) \leftarrow c_1 \wedge G_1$$
$$\ldots$$
$$\delta_m : \quad newp(X_1, ..., X_h) \leftarrow c_m \wedge G_m$$

where

- **newp is a new predicate symbol**
- $\{X_1, ..., X_h\} \subseteq FV(c_1 \wedge G_1, ..., c_m \wedge G_m)$
- **the predicates occurring in $G_1, ..., G_m$ occur also in P_0**

$Defs_n$: all definitions introduced until P_n

R2. Unfolding

By unfolding $\gamma: \; H \leftarrow c \wedge A \wedge G$

w.r.t. A we get the clauses

$$\Gamma: \; \{ H \leftarrow \overbrace{c \wedge A = H_1 \wedge c_1}^{\text{satisfiable}} \wedge G_1 \wedge G$$

for every clause $H_1 \leftarrow c_1 \wedge G_1$ in P_k }

R3. Folding

Given the clauses

$$\gamma_1 : \; H \leftarrow c \wedge c_1\vartheta \wedge G_1\vartheta \wedge G$$
$$\ldots$$
$$\gamma_m : \; H \leftarrow c \wedge c_m\vartheta \wedge G_m\vartheta \wedge G$$

and the definitions

$$\delta_1 : \; newp(X_1, ..., X_h) \leftarrow c_1 \wedge G_1$$
$$\ldots$$
$$\delta_m : \; newp(X_1, ..., X_h) \leftarrow c_m \wedge G_m$$

by folding γ_1, ..., γ_m we get the clause

$$\eta : \; H \leftarrow c \wedge newp(X_1, ..., X_h)\,\vartheta \wedge G$$

R4. Clause Removal

- **unsatisfiable constraint**

$D \models \neg\exists(c)$

γ : $H \leftarrow c \wedge G$

- **subsumed clause**

$D \models \forall(c_1 \rightarrow \exists Z c_2)$

with $Z = FV(c_2) - vars(H, G_2)$

γ : $(H \leftarrow c_1 \wedge G_1 \wedge G_2)\vartheta$

δ : $H \leftarrow c_2 \wedge G_2$

R5. Constraint Replacement

A clause $\gamma_1 : \; H \leftarrow c_1 \wedge G$

A new clause $\gamma_2 : \; H \leftarrow c_2 \wedge G$

if $D \models \forall(\exists Y c_1 \leftrightarrow \exists Z c_2)$

with $Y = FV(c_1) - vars(H,G)$ and $Z = FV(c_2) - vars(H,G)$

R6-R7. Clause Splitting and Clause Fusion

One clause

γ : $H \leftarrow (c \vee d) \wedge G$

splitting ↓ ↑ fusion

Two clauses

γ_1 : $H \leftarrow c \wedge G$

γ_2 : $H \leftarrow d \wedge G$

In particular:

true $\wedge$ ...

↓ splitting

$c \wedge$ ...

$\neg c \wedge$ ...

Correctness of the Rules

Let $P_0, ..., P_n$ be transformation sequence.

If "some conditions of performing unfolding before folding or non-recursion"

then $M(P_0 \cup Defs_n) = M(P_n)$

Determinization Strategy

a program P, a constrained atom $c \wedge p(t_1, ..., t_h)$

initial definition δ_{sp} : $p_{sp}(X_1, ..., X_r) \leftarrow c \wedge p(t_1, ..., t_h)$

Defs := $\{\delta_{sp}\}$; Cls := $\{\delta_{sp}\}$; P_{sp} := {};

while Cls≠{} do

(1) Unfold-Simplify:
clauses in Cls are unfolded and simplified --> UnfCls

(2) Partition :
clauses in UnfCls are partitioned in mutually exclusive packets --> PartCls

(3) Define-Fold :
packets in PartCls are folded introducing new definition NewDefs --> FldCls
clauses with constraints only in their bodies --> CFacts

(4) Defs:=Defs∪NewDefs; Cls:=NewDefs; P_{sp}:=P_{sp}∪CFacts∪FldCls

Specializing Leq_Match 1

leq_match(P,S) ← append(B,C,S) ∧ append(A,Q,B) ∧ leq(P,Q)
append([ ], Ys, Ys) ←
append([X|Xs], Ys, [X|Zs]) ← append(Xs, Ys, Zs)
leq([ ], [ ]) ←
leq([X|Xs], [Y|Ys]) ← X ≤ Y ∧ leq(Xs,Ys)

6. $leq_match_{sp}(S) \leftarrow leq_match([1,0,2],S)$

Unfold-Simplify clause 6 :

7. $leq_match_{sp}(S) \leftarrow append(B,C,S) \wedge append(A,Q,B) \wedge leq([1,0,2],Q)$

This clause is a packet in itself. Define :

8. $new1(S) \leftarrow append(B,C,S) \wedge append(A,Q,B) \wedge leq([1,0,2],Q)$

Fold clause 7 :

9. $leq_match_{sp}(S) \leftarrow new1(S)$

Specializing Leq_Match 2

Unfold-Simplify clause 8 :

10. new1([X|Xs]) ← 1≤X ∧ append(Q,C,Xs) ∧ leq([0,2],Q)
11. new1([X|Xs]) ← append(B,C,Xs) ∧ append(A,Q,B) ∧ leq([1,0,2],Q)

Clauses 10 and 11 are not mutually exclusive. Clause Splitting of clause 11 :

12. new1([X|Xs]) ←1≤X∧append(B,C,Xs)∧append(A,Q,B)∧leq([1,0,2],Q)
13. new1([X|Xs]) ←1>X∧append(B,C,Xs)∧append(A,Q,B)∧leq([1,0,2],Q)

Two packets of clauses : 10, 12 and 13

Specializing Leq_Match 3

Packet 10, 12 :

10. new1([X|Xs])←1≤X∧append(Q,C,Xs)∧leq([0,2],Q)
12. new1([X|Xs])←1≤X∧append(B,C,Xs)∧append(A,Q,B)∧leq([1,0,2],Q)

Folding packet 10, 12 using Definition {14, 15} :

14. new2(Xs) ← append(Q,C,Xs) ∧ leq([0,2],Q)
15. new2(Xs) ← append(B,C,Xs) ∧ append(A,Q,B) ∧ leq([1,0,2],Q)

We get :

16. new1([X|Xs]) ← 1≤X ∧ new2(Xs)

Folding packet 13 using Definition 8 we get :

17. new1([X|Xs]) ← 1>X ∧ new1(Xs)

Leq_Match 2

by specializing w.r.t. the goal leq_match([1,0,2], S)
we get the deterministic program:

9. $leq_match_{sp}(S) \leftarrow new1(S)$
16. $new1([X|Xs]) \leftarrow 1 \leq X \wedge new2(Xs)$
17. $new1([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$

$new2([X|Xs]) \leftarrow 1 \leq X \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new2([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new3([X|Xs]) \leftarrow 2 \leq X \wedge new5(Xs)$
$new3([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new3(Xs)$
$new3([X|Xs]) \leftarrow 0 \leq X \wedge 1 > X \wedge new4(Xs)$
$new3([X|Xs]) \leftarrow 0 > X \wedge new1(Xs)$
$new4([X|Xs]) \leftarrow 2 \leq X \wedge new6(Xs)$
$new4([X|Xs]) \leftarrow 1 \leq X \wedge 2 > X \wedge new2(Xs)$
$new4([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$
$new5(Xs) \leftarrow$
$new6(Xs) \leftarrow$

Program Synthesis

Given a locally stratified CLP(*D*) program P and

a first order formula φ[X]

Derive an efficient program S defining s(X) s.t.

∀ ground term t

M(P) ⊨φ[t] iff M(S) s(t)

---

1. Apply Lloyd-Topor transformation starting from :

$$s(X) \leftarrow \varphi[X]$$

and derive a possibly inefficient, locally stratified program Cls(s,φ)
s.t. ∀ ground term t, M(P) φ[t]⊨ iff M(P ∪ Cls(s,φ)) s(t) ⊨

2. Apply rules according to a strategy: $P \cup Cls(s,\varphi) \xrightarrow{\text{rules}}{}^{*} S$

Synthesis of max 1

Program P :

1. list([ ]) ←
2. list([X|Xs]) ← list(Xs)
3. member(X, [A|As]) ← X = A
4. member(X, [A|As]) ← member(X,As)

$\varphi[X]$: list(L) $\wedge$ member(M,L) $\wedge$ $\forall$X (member(X,L) $\rightarrow$ X$\leq$M)

We start from :

max(L,M) ← list(L) $\wedge$ member(M,L) $\wedge\forall$X(member(X,L) $\rightarrow$ X$\leq$M)

After Lloyd-Topor we get :

6. max(L,M) ← list(L) $\wedge$ member(M,L) $\wedge$ $\neg$ new1(L,M)
7. max(L,M) ← member(X,L) $\wedge$ $\neg$ X$\leq$M

Synthesis of max 2

Synthesized Program :

16. max([A|As], M) ← new2(A, As, M)
17. new2(A, [ ], M) ← M = A
21. new2(A, [B|As], M) ← B ≤ A ∧ new2(A, As, M)
22. new2(A, [B|As], M) ← A ≤ B ∧ new2(B, As, M)

Program Verification

Given a locally stratified CLP(*D*) program P and

a first order formula φ

Check whether or not $M(P) \models \varphi$

---

1. Apply Lloyd-Topor transformation starting from :

$f \leftarrow \varphi$

and derive a possibly inefficient, locally stratified program Cls(f,φ)

s.t. $M(P) \not\models \varphi$ iff $M(P \cup Cls(f,\varphi)) \not\models f$

2. Apply rules according to a strategy: $P \cup Cls(f,\varphi) \xrightarrow{\text{rules}} T^*$

s.t. either $f \leftarrow$ is in T (in which case $M(P) \models \varphi$)

or no clause for f is in T (in which case $M(P) \not\models \varphi$)

Future Work

Specialization:

- Experiment with other classes of algorithms : scheduling
- Adding a generalization technique to get termination for general classes of programs

Verification:

- Experiment with other protocols and properties
- Looking for complete strategies w.r.t. some given classes of programs

References

Lopstr 2000 = LNCS 2042

CL 2000 = LNAI 1861

Kowalski Festschrift 2002 = LNAI 2407

Fabio's Thesis 2002

AGP 2002

Visit :

`http://www.iasi.rm.cnr.it/~proietti`

MAP Transformationn System available at :

`http://www.iasi.rm.cnr.it/~fioravan`

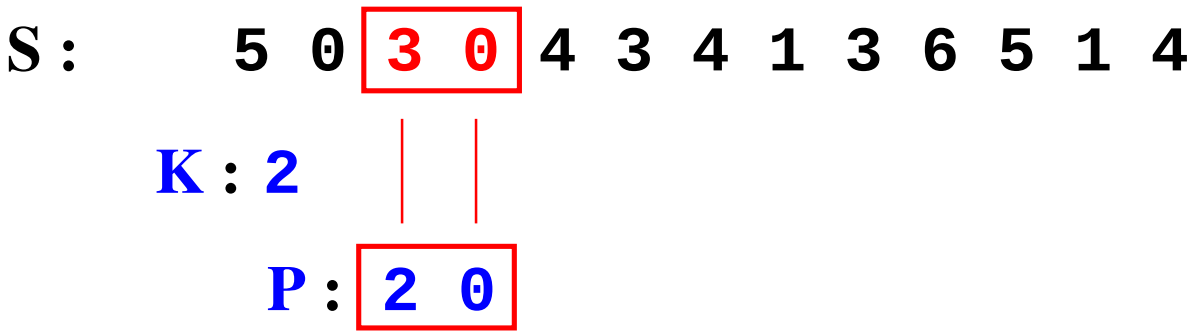
More slides on specialization (2)

Near_Match 1

near_match(P,K,S) ← app(B,C,S) ∧ app(A,Q,B) ∧ near(P,K,Q)

app([ ], Ys, Ys) ←
app([X|Xs], Ys, [X|Zs]) ← app(Xs, Ys, Zs)

near([ ],K,[ ]) ←
near([X|Xs], K, [Y|Ys]) ← |X-Y| ≤ K ∧ leq(Xs,Ys)

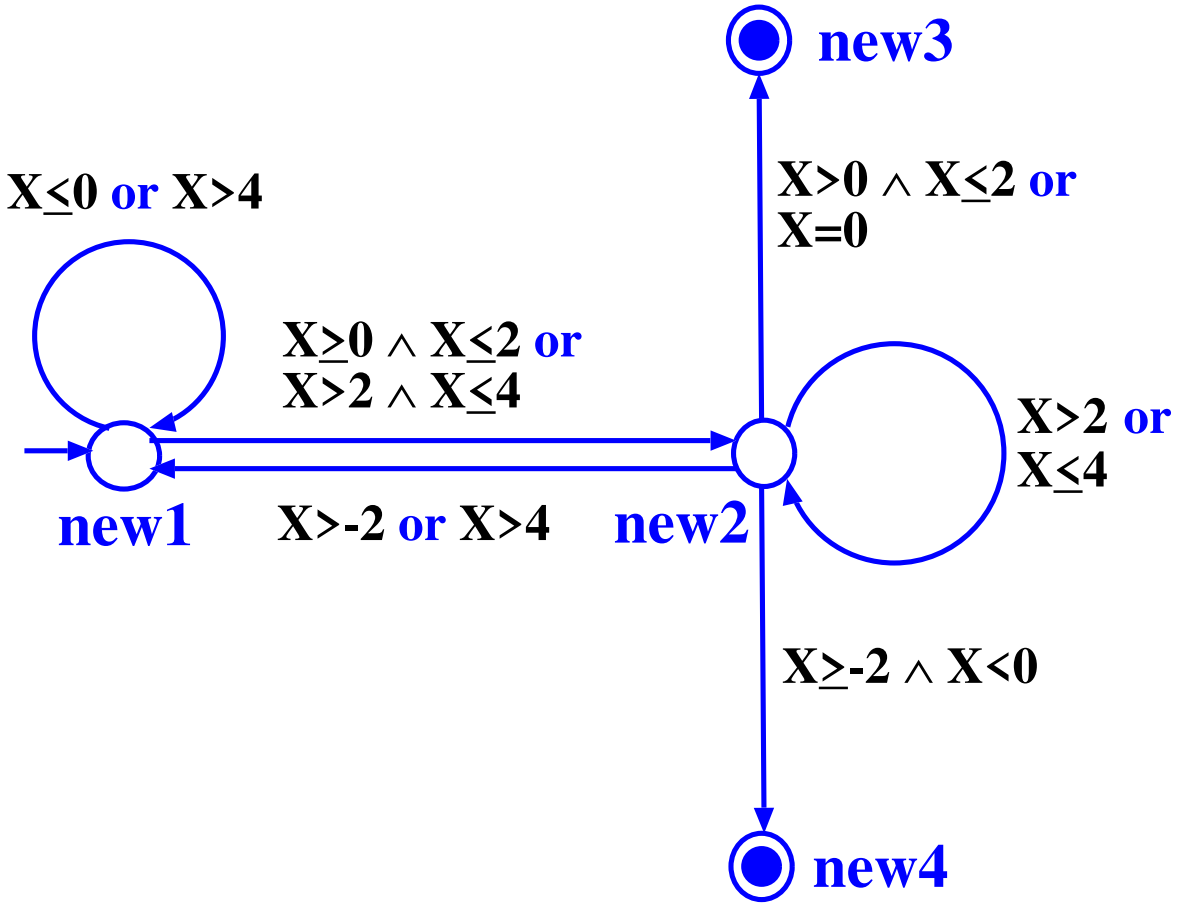


Near_Match 2

by specializing near_match w.r.t. the goal near_match([2,0], 2, S) we get the deterministic program:

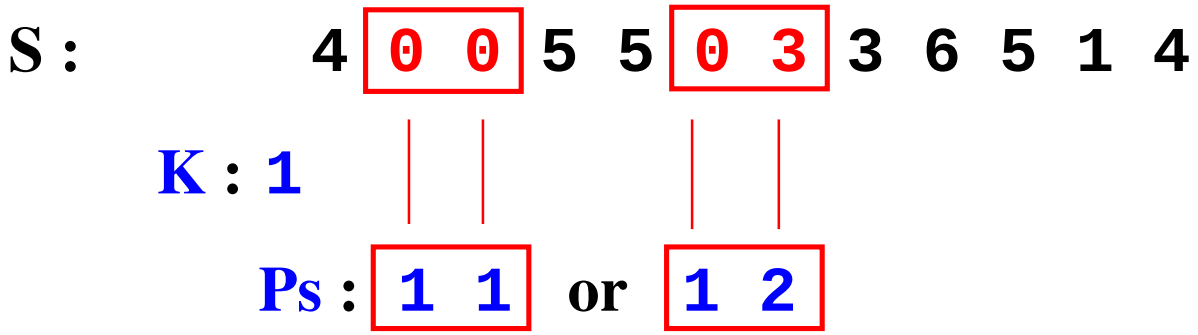
$near_match_{sp}(S) \leftarrow new1(S)$
$new1([X|Xs]) \leftarrow X \geq 0 \wedge X \leq 2 \wedge new2(Xs)$
$new1([X|Xs]) \leftarrow X \leq 0 \wedge new1(Xs)$
$new1([X|Xs]) \leftarrow X > 2 \wedge X \leq 4 \wedge new2(Xs)$
$new1([X|Xs]) \leftarrow X > 4 \wedge new1(Xs)$
$new2([X|Xs]) \leftarrow X > 2 \wedge X \leq 4 \wedge new2(Xs)$
$new2([X|Xs]) \leftarrow X > 0 \wedge X \leq 2 \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow X = 0 \wedge new3(Xs)$
$new2([X|Xs]) \leftarrow X \geq -2 \wedge X < 0 \wedge new4(Xs)$
$new2([X|Xs]) \leftarrow X < -2 \wedge new1(Xs)$
$new2([X|Xs]) \leftarrow X > 4 \wedge new1(Xs)$
$new3(Xs) \leftarrow$
$new4(Xs) \leftarrow$

Near_Match 3


new3
X≤0 or X>4
X>0 ∧ X≤2 or
X=0
X≥0 ∧ X≤2 or
X>2 ∧ X≤4
X>2 or
X≤4
new1
X>-2 or X>4
new2
X≥-2 ∧ X<0
new4

Near_multimatch

near_mmatch([P|Ps], K, S) ← near_match(P, K, S)
near_mmatch([P|Ps], K, S) ← near_mmatch(Ps, K, S)



Interval_Regular_Expressions 1

in_language(E,S) ← string(S) ∧ accepts(E,S)

accepts(<(X), [Y]) ← Y<X — less than

accepts(>(X), [Y]) ← Y>X — greater than

accepts(int(X_1, X_2),[Y]) ← X_1<Y ∧ Y<X_2 — within interval

accepts(E_1·E_2, S) ← app(S_1,S_2,S) ∧ accepts(E_1, S_1) ∧ accepts(E_2, S_2)

accepts(E_1+E_2, S) ← accepts(E_1,S)

accepts(E_1+E_2, S) ← accepts(E_2,S)

accepts(E*, [ ]) ←

append for nonempty lists

accepts(E*, S) ← ne_app(S_1,S_2,S) ∧ accepts(E, S_1) ∧ accepts(E*,S_2)

string([ ]) ←

string([X|Xs]) ← string(Xs)

Interval_Regular_Expressions 2

E : int(0,2)* + int(1,3)*

S1 : 1 2 2 1 2 2 2 1 3 1

in_language(E, S1) holds

S2 : 1 2 2 1 2 2 2 4 0 1

in_language(E, S2) does not hold

E : (>1) · (>1)* + (>1) · (>1)* · (<0)

S3 : 3 2 4 3 3 2 2 3 3 2

in_language(E, S3) holds

S4 : 3 2 4 3 3 2 2 3 3 1

in_language(E, S4) does not hold

Speedups

Program	Constrained Atom for Specialization	Speedup
Leq_Match	leq_match([1,0,2], S)	2.7
Near_Match	near_match([2,0], 2, S)	37-56
Near_Match	near_match([2,0,4], 2, S)	46
Near_Mmatch	near_mmatch([[1,1],[1,2]], 1, S)	45
Int_Reg_Expr	in_language(int(0,2)*+int(1,3)*, S)	1.2-7
Int_Reg_Expr	in_language((>1)(>1)*+(>1)(>1)*(<0), S) . .	1.3-320

Other slides

Correctness of the Determinization Strategy

Given a program P, if the Determinization Strategy with clause

δ_{sp} : $p_{sp}(X_1, ..., X_r) \leftarrow c \wedge p(t_1, ..., t_h)$

returns P_{sp}

then P_{sp} **is** **deterministic**

Specializing a Matcher (2)

By unfolding new3([X|S]) ← X≠b, new2([X|S])

using
new2([a|S]) ← new3(S)
new2([X|S]) ← X≠a, new1(S)

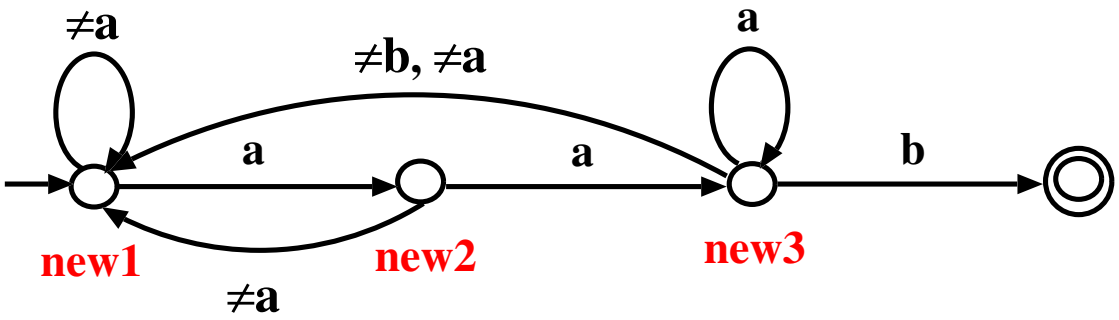
we get:

string S →

$match_S$(S) ← new1(S)
new1([a|S]) ← new2(S)
new1([X|S]) ← X≠a, new1(S)
new2([a|S]) ← new3(S)
new2([X|S]) ← X≠a, new1(S)
new3([b|S]) ←
new3([a|S]) ← new3(S)
new3([X|S]) ← X≠b, X≠a, new1(S)

→ true/false

Finite automaton (Knuth-Morris-Pratt) :



- Clauses correspond to transitions and are mutually exclusive
- Predicates correspond to states