An Overview of Program Transformation

Maurizio Proietti

IASI-CNR
Viale Manzoni 30
00185 Rome, Italy
http://www.iasi.cnr.it/~proietti

Joint work with:
A.Pettorossi, F.Fioravanti, S.Renault, V.Senni

IASI - MoMo Day **17 Ottobre 2005**

Program Transformation

- Theory transformation: Changing the axioms of a theory, preserving its models.

a=b b=c	a=b b=c a=c

- Program transformation: Changing the text of a program, preserving its semantics (and improving efficiency).

- Syntax — Semantics

Logic programs	least Herbrand model, perfect model, ...
Functional programs	call-by-value, call-by-name, ...
...	

Plan

Rule-Based Program Transformation

Transformation Rules

Transformation Strategies

Applications of Program Transformation

1. Program improvement:
   from slow programs to fast programs

2. Program synthesis:
   from specifications to programs

3. Program verification:
   from programs & formulas to true/false

Rule-based Program Transformation

[Burstall-Darlington 77]

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

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

- Each rule is a local transformation of a program P_i into a new program P_{i+1}
- Each rule application preserves the semantics M:

$$M(P_i)=M(P_{i+1})$$

- The application of the rules is guided by a strategy

Transformation Rules

Unfolding Rule

Applying an SLD-resolution step ("backward chaining")

$q \leftarrow r$
$q \leftarrow s$
$p \leftarrow q$ ⟶ $q \leftarrow r$, $q \leftarrow s$, $p \leftarrow r$, $p \leftarrow s$ replace head by bodies

$q(f(a))$
$q(b)$
$q(f(b)) \leftarrow r(b)$
$p(X) \leftarrow q(f(X))$ ⟶ $q(f(a))$, $q(b)$, $q(f(b)) \leftarrow r(b)$, $p(a)$ unify +, $p(b) \leftarrow r(b)$ replace head by bodies

Folding Rule

Inverse of unfolding

$q \leftarrow r$
$q \leftarrow s$
$p \leftarrow r$
$p \leftarrow s$

$q \leftarrow r$
$q \leftarrow s$
$p \leftarrow q$

replace bodies by head

instance of

$s(X) \leftarrow q(X) \wedge r(X)$
$p(X) \leftarrow q(f(X)) \wedge r(f(X))$

$s(X) \leftarrow q(X) \wedge r(X)$
$p(X) \leftarrow s(f(X))$

replace instance of body by head

Clause removal, Goal removal

~~$q \leftarrow q$~~ $q \leftarrow q$ is true

~~$p \leftarrow q \wedge \neg r$~~ $\neg r$ is false, $p \leftarrow q \wedge$ false is true
r

$p \leftarrow q$ $p \leftarrow q$ implies $p \leftarrow q \wedge r$
~~$p \leftarrow q \wedge r$~~

$p \leftarrow$ ~~$\neg r$~~ $\neg r$ is true
$r \leftarrow$ false

$p \leftarrow q \wedge$ ~~q~~

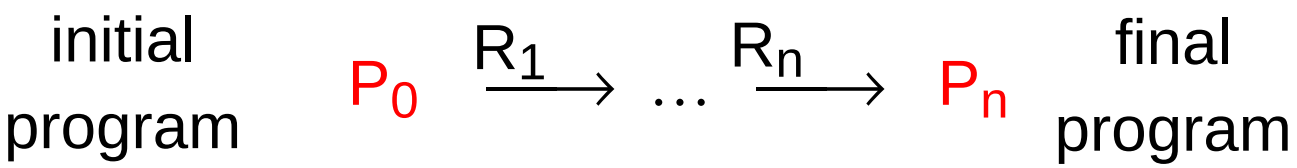
Definition Introduction

Introducing a new predicate

q(a)
q(b)
r(b)

q(a)
q(b)
r(b)
$newp(X) \leftarrow q(X) \wedge \neg r(X)$

Correctness of the Rules



Theorem:

If $R_i \in$ {Definition, Unfolding, Folding, Clause Removal, Goal Removal} and every definition is unfolded then

$$M(P_0 \cup Defs) = M(P_n)$$

where Defs are the new definitions introduced during transformation and M(P) is the least Herbrand (perfect) model of P.

Limitations of Program Transformation

There exists no complete set of rules such that

if M(P) = M(Q) then $P \xrightarrow{R_1} \ldots \xrightarrow{R_n} Q$

Transformation Strategies

Need for Strategies

unfolding ; folding = identity

Transformation Strategies for Program Improvement

Syntax-directed strategies can be automated for classes of programs

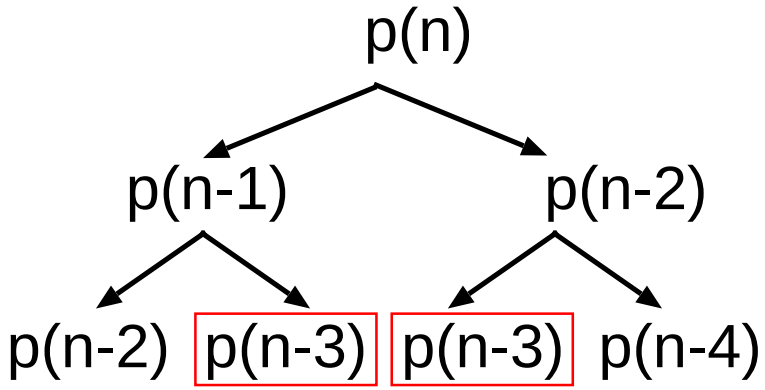
- Deriving linear recursive programs: avoiding multiple recursive calls with common subcalls
- Deriving tail recursive programs: avoiding the use of a stack to implement recursion
- Eliminating existential variables: avoiding the computation of unnecessary values
- Specialization: adapting programs to the context of use
- Determinization: avoiding nondeterministic search

An Example of Exponential Speedup (1)

from binary recursion to linear recursion

p(0)
p(s(0))
$p(s(s(X))) \leftarrow p(s(X)) \wedge p(X)$ binary recursion

n stands for $s^n(0)$



p(n) is proved in 2^n steps

An Example of Exponential Speedup (2)

$p(0)$

$p(s(0))$

$p(s(s(X))) \leftarrow p(s(X)) \wedge p(X)$

definition

$q(Y) \leftarrow p(s(Y)) \wedge p(Y)$

An Example of Exponential Speedup (3)

$p(0)$

$p(s(0))$

$p(s(s(X))) \leftarrow p(s(X)) \wedge p(X)$

definition

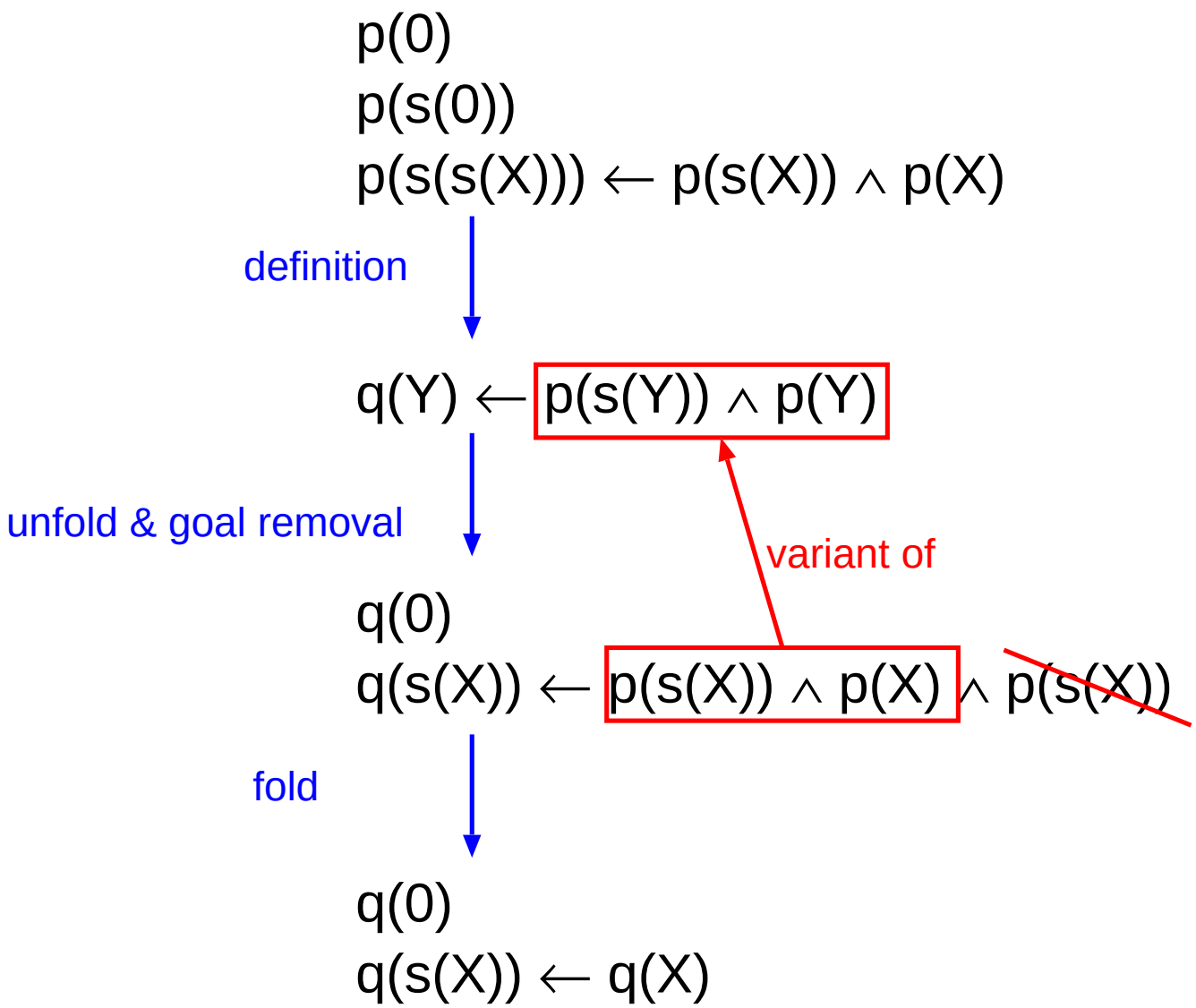
$q(Y) \leftarrow p(s(Y)) \wedge p(Y)$

unfold & goal removal

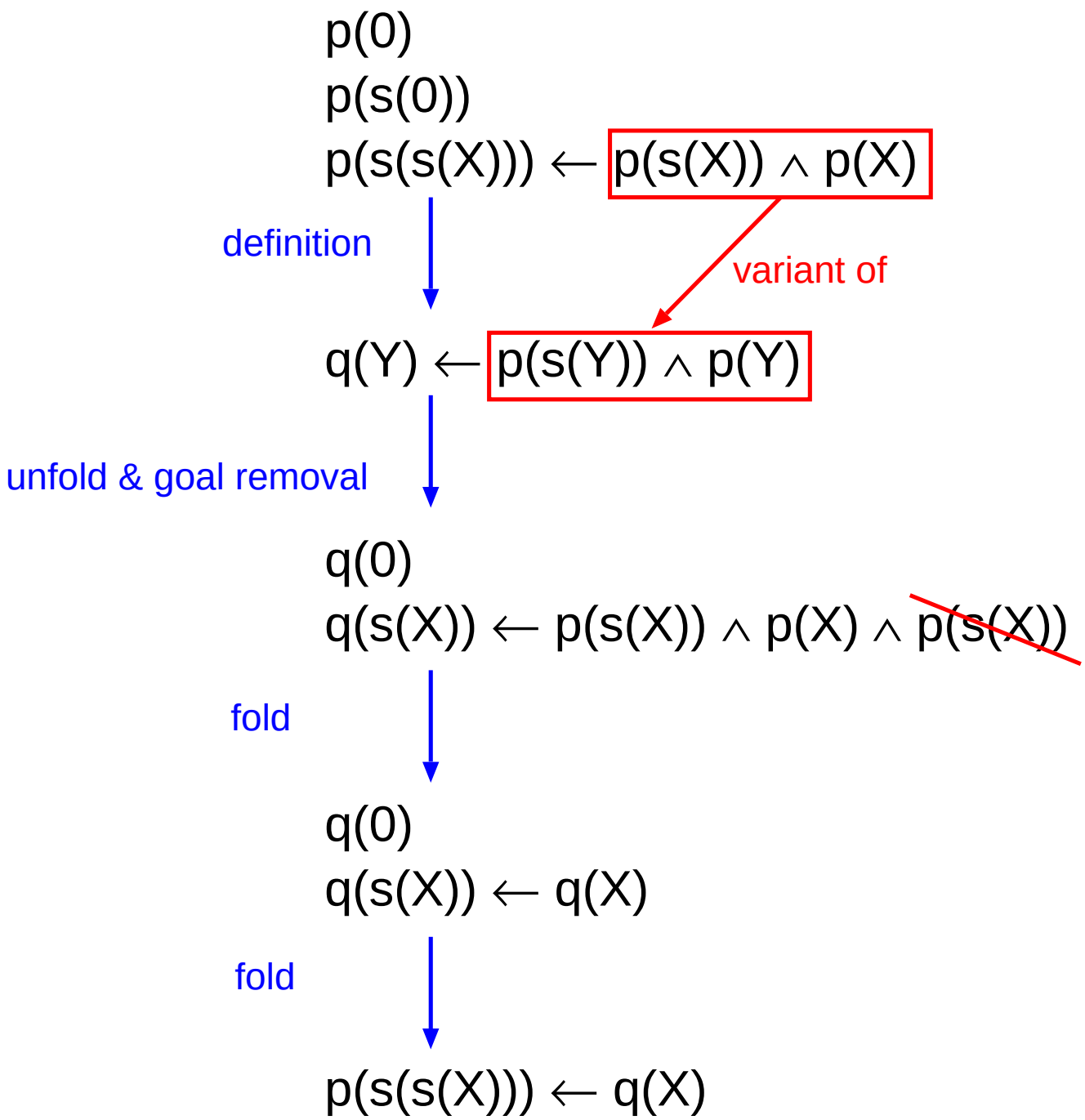
(Y=0) $q(0)$

(Y=s(X)) $q(s(X)) \leftarrow p(s(X)) \wedge p(X) \wedge \cancel{p(s(X))}$

An Example of Exponential Speedup (4)



An Example of Exponential Speedup (5)



An Example of Exponential Speedup (6)

Final program:

p(0)

p(s(0))

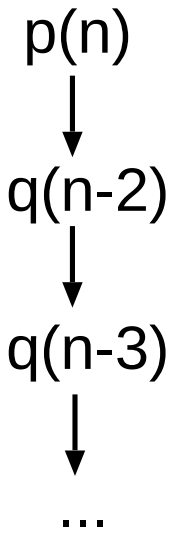
p(s(s(X))) ← q(X)

q(0)

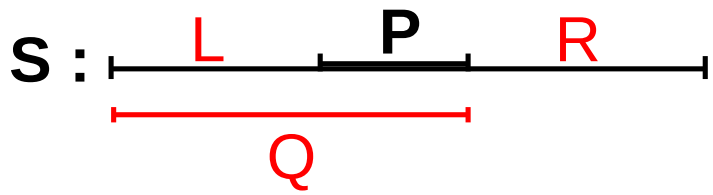
q(s(X)) ← q(X)

linear recursion

p(n) is proved in n-1 steps



Eliminating Existential Variables (1)



$\exists$ Q, L, R (L :: P = Q $\wedge$ Q :: R = S)

Unnecessary existential variables

match(P,S) $\leftarrow$ append(L,P,Q) $\wedge$ append(Q,R,S)

append([],X,X)

append([A|X],Y,[A|Z]) $\leftarrow$ append(X,Y,Z)

We can improve the match program by eliminating existential variables

Eliminating Existential Variables (2)

match(P,S) ← append(L,P,Q) ∧ append(Q,R,S)

Unfold

append([],Y,Y)

append([A|X],Y,[A|Z]) ← append(X,Y,Z)

match(P,S) ← append(P,R,S)

match(P,S) ← append(X,P,Z) ∧ append([A|Z],R,S)

Eliminating Existential Variables (3)

match(P,S) ← append(L,P,Q) ∧ append(Q,R,S)

Unfold

append([],Y,Y)

append([A|X],Y,[A|Z]) ← append(X,Y,Z)

match(P,S) ← append(P,R,S)

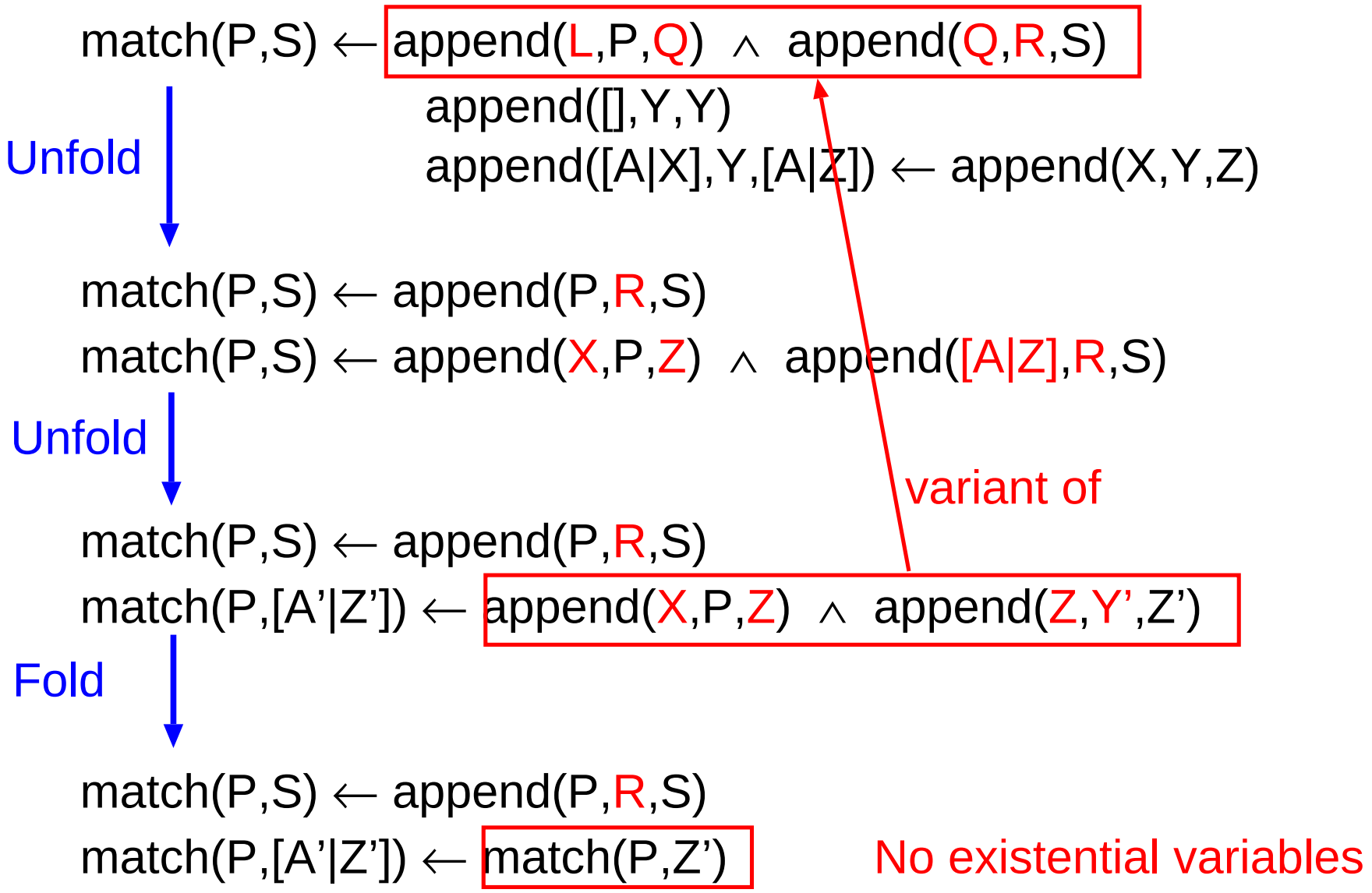
match(P,S) ← append(X,P,Z) ∧ append([A|Z],R,S)

Unfold

match(P,S) ← append(P,R,S)

match(P,[A'|Z']) ← append(X,P,Z) ∧ append(Z,Y',Z')

Eliminating Existential Variables (4)



Eliminating Existential Variables (5)

match(P,S) ← append(P,R,S)

Introduce a new definition and fold

match(P,S) ← prefix(P,S)

prefix(P,S) ← append(P,R,S)

No existential variables

unfold; fold

prefix([],S)

prefix([A|X],[A|Y]) ← prefix(X,Y)

No existential variables

Match Program Without Existential Variables

match(P,S) ← prefix(P,S)

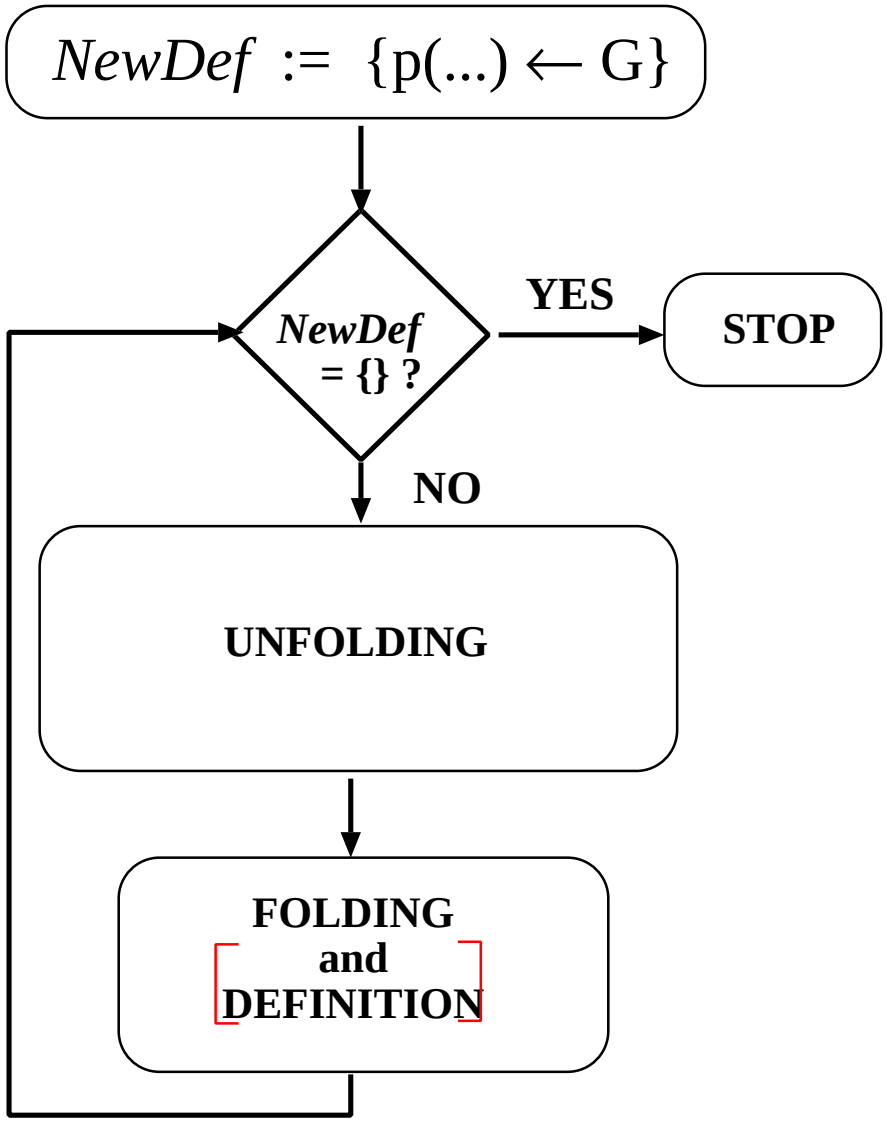
match(P,[A'|Z']) ← match(P,Z')

prefix([],S)

prefix([A|X],[A|Y]) ← prefix(X,Y)

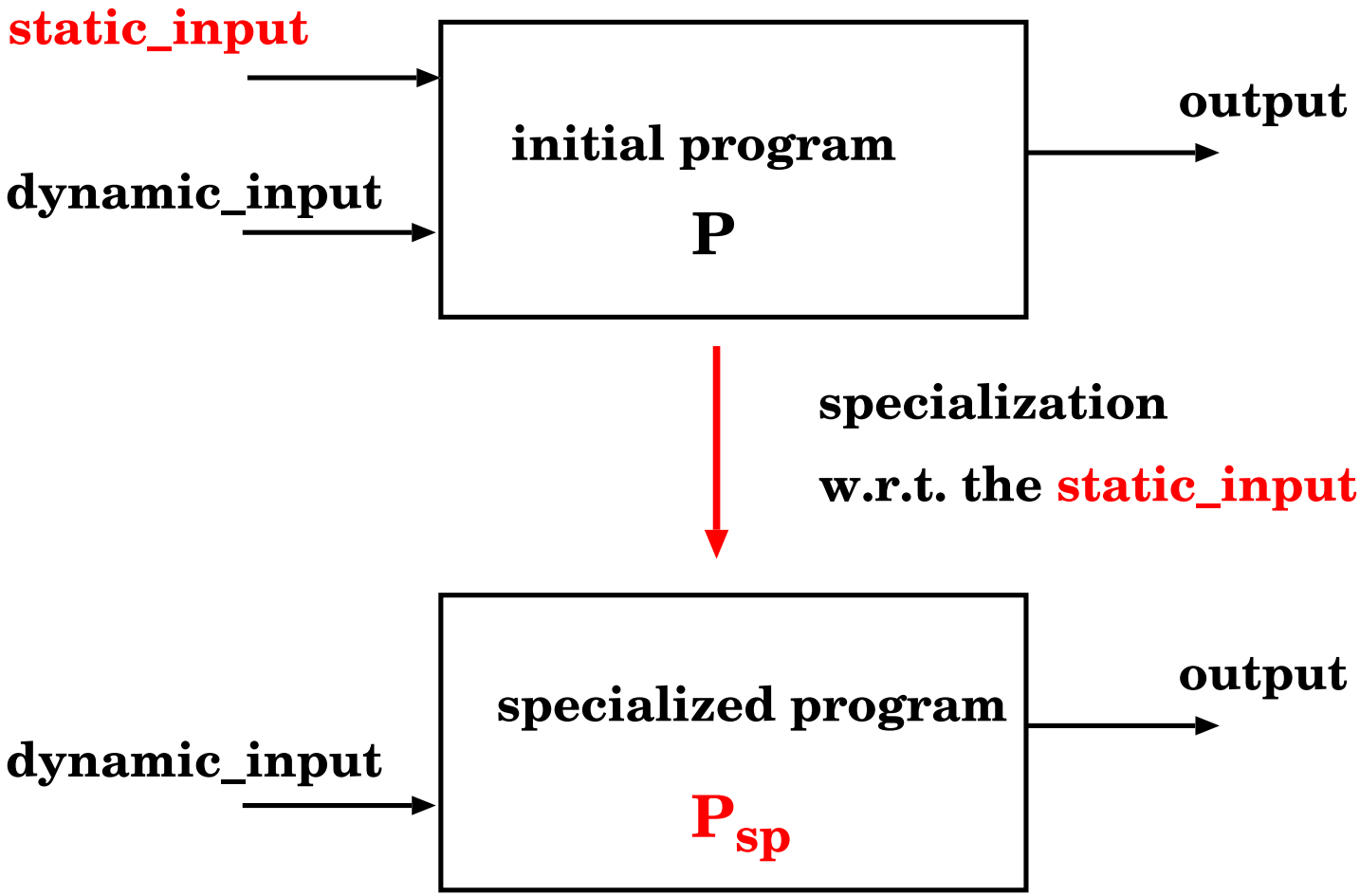
No intermediate list. Nondeterminism has been reduced

General Form of Transformation Strategy

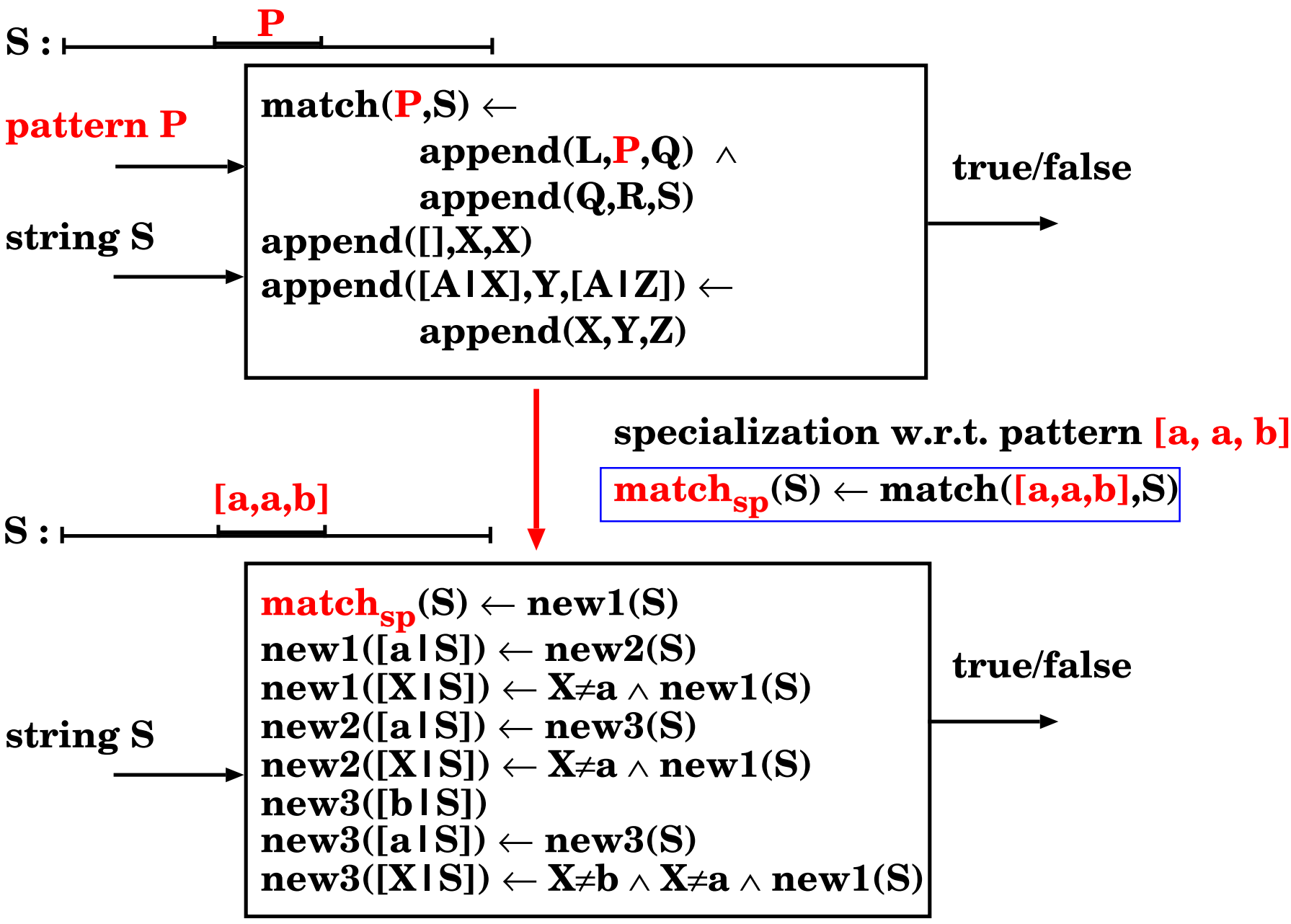


Program Improvement via Specialization

Program Specialization



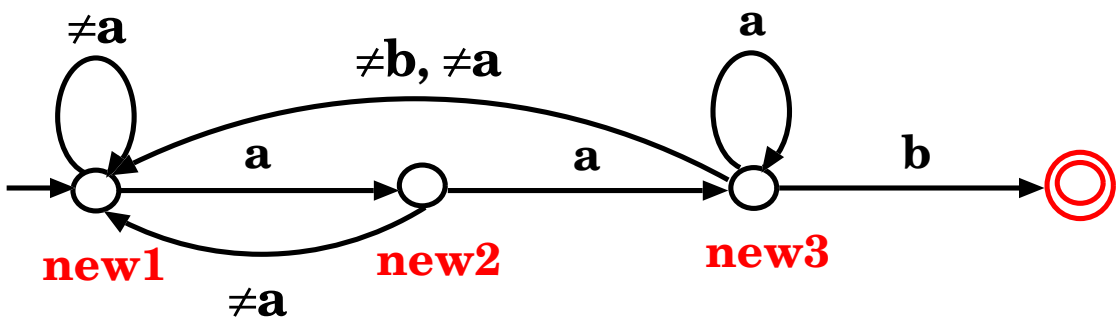
Specializing a Matcher (1)



Specializing a Matcher (2)

$\mathbf{match_s(S) \leftarrow new1(S)}$
$\mathbf{new1([a|S]) \leftarrow new2(S)}$
$\mathbf{new1([X|S]) \leftarrow X{\neq}a \wedge new1(S)}$
$\mathbf{new2([a|S]) \leftarrow new3(S)}$
$\mathbf{new2([X|S]) \leftarrow X{\neq}a \wedge new1(S)}$
$\mathbf{new3([b|S])}$
$\mathbf{new3([a|S]) \leftarrow new3(S)}$
$\mathbf{new3([X|S]) \leftarrow X{\neq}b \wedge X{\neq}a \wedge new1(S)}$

The specialized program corresponds to a DFA (~ Knuth-Morris-Pratt algorithm)



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

Other Examples of Program Specialization

Multimatch (sp wrt a set of patterns)

Membership to regular language (sp wrt a regular expression)

Matching a regular expression (sp wrt a re)

Parsing a context free parser (sp wrt a regular grammar)

Constrained matching (e.g. ’near’ match)

Experiments Using the MAP System

Static input	Specialization Time (sec)	Speedup
match([aab],S)	0.07	6.8 x 10^3
mmatch([[aaa],[aab]],S,N)	0.28	6.2 x 10^3
in_language((aa*(b+bb))*,S)	0.42	3.9 x 10^5
re_match(aa*b,S)	0.21	3.0 x 10^6
string_parse(g,[s],W)	1.62	87.1
near_match([2,0,4], 2, S)	1.89	46

Program Synthesis via Transformation

The Unfold/Fold Synthesis Method

Given a specification of a predicate s of the form:

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

where $\varphi[X]$ is a first order formula with free variable X,
derive an efficient program S which computes s

- Step 1. Generate a specification of s in clausal form (Lloyd-Topor transformation), that is, a (possibly inefficient) logic program:

$$Q: \quad s(X) \leftarrow G$$
$$\ldots$$

- Step 2. Apply the unfold/fold transformation strategy

$$Q \rightarrow \ldots \rightarrow S$$

Example: Maximum of a Set

Specification of max:

$\max(S,M) \leftrightarrow M \in S \wedge \neg\exists N\ (N \in S \wedge N > M)$

where:

$0 \in [1|S]$

$s(N) \in [B|S] \leftarrow N \in S$

$s(N) > 0$

$s(N) > s(M) \leftarrow N > M$

A set of natural numbers is represented by a list of 0s and 1s.

$\{0,1,3\} \longrightarrow [1,1,0,1,0,0]$

Maximum of a Set, Clausal Form

Step 1. Lloyd-Topor transformation:

Multiple variables

$$\max(S,M) \leftarrow M \in S \wedge \neg \text{exists\_greater}(S,M)$$

$$\text{exists\_greater}(S,M) \leftarrow N \in S \wedge N > M$$

Existential variables

Inefficient, generate-and-test program.

Maximum of a Set, Transformed

Step 2: Unfold/fold program transformation strategy.

max([1|S],0) ← empty(S)
max([B|S],s(M)) ← max(S,M)
empty([ ])
empty([0|S]) ← empty(S)

No multiple variables.
No existential variables.
Deterministic, linear time, when both arguments are ground terms

Efficiency Improvements

- Weak monadic second order theory of 1 successor (WS1S) [Buchi '60]

$$\varphi ::= N_1 > N_2 \mid N_1 = N_2 \mid N \in S \mid S_1 = S_2 \mid$$
$$\neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists N\, \varphi \mid \exists S\, \varphi$$

- WS1S expresses properties of the membership of finite sets of natural numbers

- WS1S is decidable. The decision problem for WS1S has inherent complexity $2^{2^{\iddots^{2^{n}}}}$ (d twos) for some d>0.

- For every WS1S formula the unfolf/fold transformation strategy synthesizes a program with O(n) time complexity.

The Unfold/Fold Verification Method

The Unfold/Fold Verification Method

Given a program P and a closed first order formula φ,
check whether $M(P) \models \varphi$ or not.

- Step 0. Consider the property prop specified as

$$prop \leftrightarrow \varphi$$

- Step 1. Generate a specification of prop in clausal form (Lloyd-Topor transformation):

$$Q: \quad prop \leftarrow G$$
$$\ldots$$

- Step 2. Apply the unfold/fold transformation strategy

$$Q \rightarrow \ldots \rightarrow S$$

If S contains the fact prop then $M(P) \models \varphi$
If S contains no clauses for prop then $M(P) \models \neg\varphi$

Verification of max (1)

Every non-empty set has a maximum element

P: $0 \in [1|S]$

$s(N) \in [B|S] \leftarrow N \in S$

$s(N) > 0$

$s(N) > s(M) \leftarrow N > M$

To be verified:

$M(P) \models \forall S((\exists X\ X \in S) \rightarrow \exists M\ (M \in S \wedge \neg \exists N\ (N \in S \wedge N > M))$

Verification of max (2)

$prop \leftrightarrow \forall S((\exists X\ X \in S) \rightarrow \exists M\ (M \in S \wedge \neg\exists N\ (N \in S \wedge N > M))$

Lloyd-Topor transformation

$prop \leftarrow \neg new1$

$new1 \leftarrow X \in S \wedge \neg new2(S)$

$new2(S) \leftarrow M \in S \wedge \neg new3(S,M)$

$new3(S,M) \leftarrow N \in S \wedge N > M$

Elimination of existential variables

$prop \leftarrow \neg new1$

$new1 \leftarrow new4$

$new1 \leftarrow new1$

$new4 \leftarrow new4$

Propositional program

Verification of max (3)

$prop \leftrightarrow \forall S((\exists X\ X \in S) \rightarrow \exists M\ (M \in S \wedge \neg\exists N\ (N \in S \wedge N > M))$

Lloyd-Topor transformation

$prop \leftarrow \neg new1$

$new1 \leftarrow X \in S \wedge \neg new2(S)$

$new2(S) \leftarrow M \in S \wedge \neg new3(S,M)$

$new3(S,M) \leftarrow N \in S \wedge N > M$

Elimination of existential variables

$prop \leftarrow \neg new1$

$new1 \leftarrow new4$

~~$new1 \leftarrow new1$~~

~~$new4 \leftarrow new4$~~

tautologies

Verification of max (4)

$prop \leftrightarrow \forall S((\exists X\ X \in S) \rightarrow \exists M\ (M \in S \wedge \neg\exists N\ (N \in S \wedge N > M))$

Lloyd-Topor transformation

$prop \leftarrow \neg new1$

$new1 \leftarrow X \in S \wedge \neg new2(S)$

$new2(S) \leftarrow M \in S \wedge \neg new3(S,M)$

$new3(S,M) \leftarrow N \in S \wedge N > M$

Elimination of existential variables

$prop \leftarrow \neg new1$

~~$new1 \leftarrow new4$~~ new4 is false (no clauses for new4)

Verification of max (5)

$prop \leftrightarrow \forall S((\exists X\ X \in S) \rightarrow \exists M\ (M \in S \wedge \neg\exists N\ (N \in S \wedge N > M))$

Lloyd-Topor transformation

$prop \leftarrow \neg new1$

$new1 \leftarrow X \in S \wedge \neg new2(S)$

$new2(S) \leftarrow M \in S \wedge \neg new3(S,M)$

$new3(S,M) \leftarrow N \in S \wedge N > M$

Elimination of existential variables

$prop \leftarrow \neg new1$

new1 is false (no clauses for new1)

Verification of max (6)

$prop \leftrightarrow \forall S((\exists X\ X \in S) \rightarrow \exists M\ (M \in S \wedge \neg \exists N\ (N \in S \wedge N > M))$

Lloyd-Topor transformation

$prop \leftarrow \neg new1$

$new1 \leftarrow X \in S \wedge \neg new2(S)$

$new2(S) \leftarrow M \in S \wedge \neg new3(S,M)$

$new3(S,M) \leftarrow N \in S \wedge N > M$

Elimination of existential variables

$prop$ prop is true in M(P)

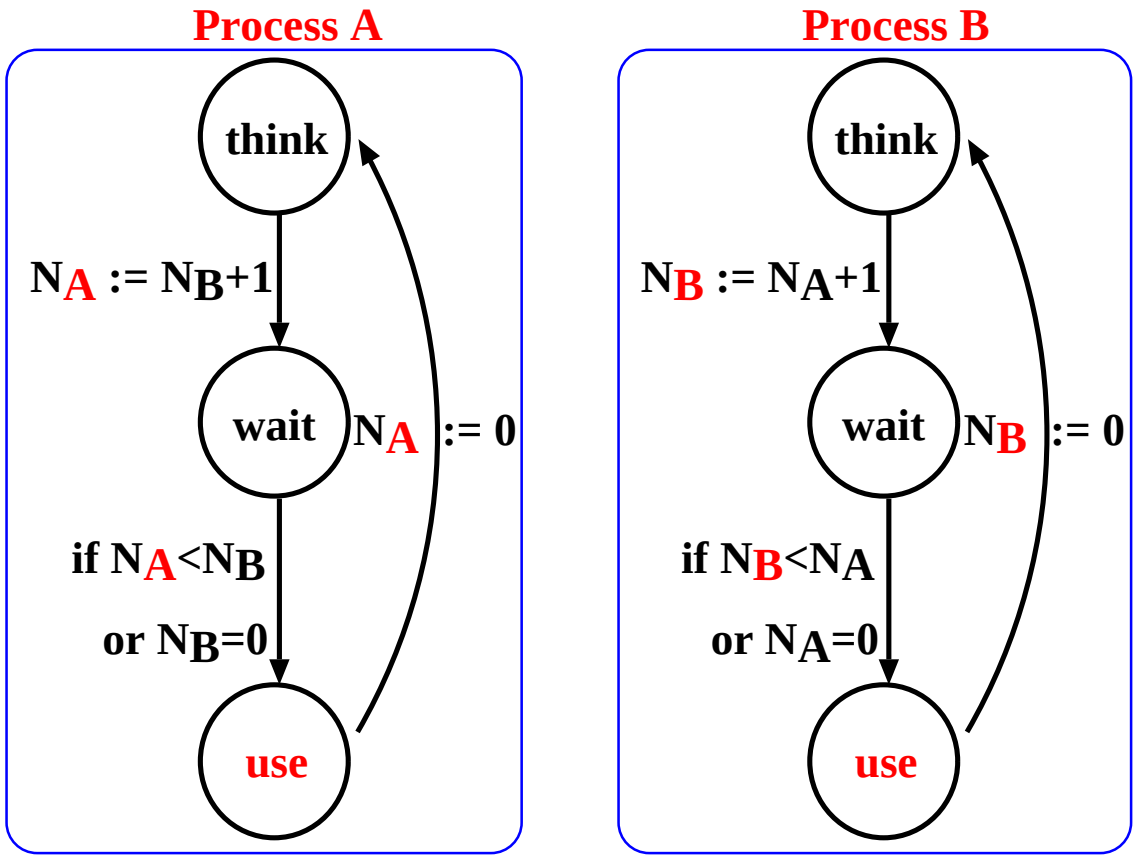
In general, WS1S can be decided by the unfold/fold method

Protocol Verification

The unfold/fold verification method can be used for the verification of infinite state systems (unbounded state space).

Examples: Mutual exclusion protocols using an integer counter, Petri nets, parameterized cache coherence protocols, ...

Bakery Protocol [Lamport]



Experimental Results using the MAP System

Mutual exclusion

Reachability in Petri nets

Cache coherence

System	Prop	Time (sec)
Bakery	safety	0,12
	liveness	0.34
Ticket	safety	0.52
	liveness	1.00
ResetPetriNet	reach1	0.06
	reach2	0.06
	reach3	0.05
Berkeley RISC	safety1	1.14
	safety2	1.14
Xerox Dragon	safety1	0.67
	safety2	0.68
	safety3	0.67
DEC Firefly	safety1	0.23
	safety2	0.24
IEEE Futurebus	safety	0.95
Illinois Univ.	safety1	0.25
	safety2	0.25
MESI	safety1	0.24
	safety2	0.21
MOESI	safety1	1.11
	safety2	1.09
Synapse	safety1	0.14
	safety2	0.14