Formal Aspects of Program Transformation: Rules and Strategies

Alberto Pettorossi

DISP - Università di Roma Tor Vergata
Via del Politecnico 1
00133 Rome, Italy
http://www.iasi.cnr.it/~adp

Maurizio Proietti

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

Dagstuhl, Germany
April 18-22, 2005

- **Transformation:** 1. from "slow" programs to "fast" programs
  (preserving semantics and improving efficiency)
  program improvement

  2. from specifications to programs
  (preserving semantics) program synthesis

  3. from "programs & formulas" to true/false
  program verification

- "Small" specifications, "small" programs, and "small" formulas
  (discrete mathematics, optimization, combinatorial problems, searching, sorting, protocols, etc.)

- **Techniques based on the syntax of the language:**
  - **functional programs**
  - **logic programs**
  - imperative programs
  - concurrent programs

based on theorem proving ideas:

- **rules**
- **strategies**

1. The Unfold/Fold Improvement Method

Program Transformation (Burstall-Darlington)

```
fib(0) = 0
fib(1) = 1
fib(n+2) = fib(n+1)+ fib(n)
```

exponential time

transformation rules

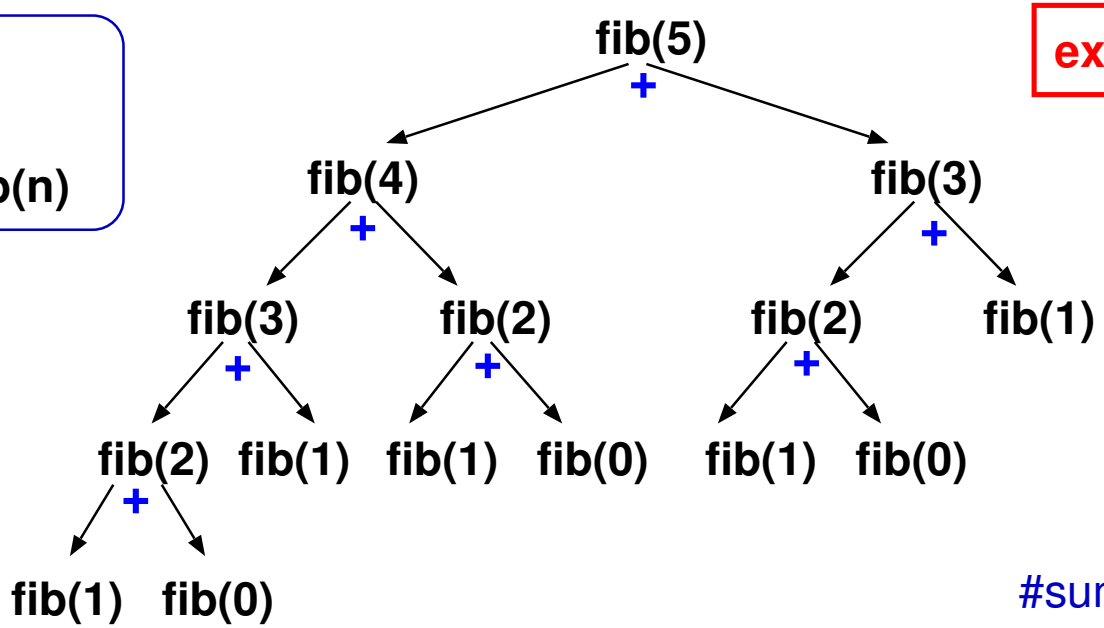
```
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(n+3) = 2u+v     where <u,v> = g(n)
g(0) = <1,0>
g(n+1) = <u+v, u>  where <u,v> = g(n)
```

linear time

R.M. Burstall, J. Darlington: "A Transformation System for Developing Recursive Programs" Journ. ACM. Jan. 1977

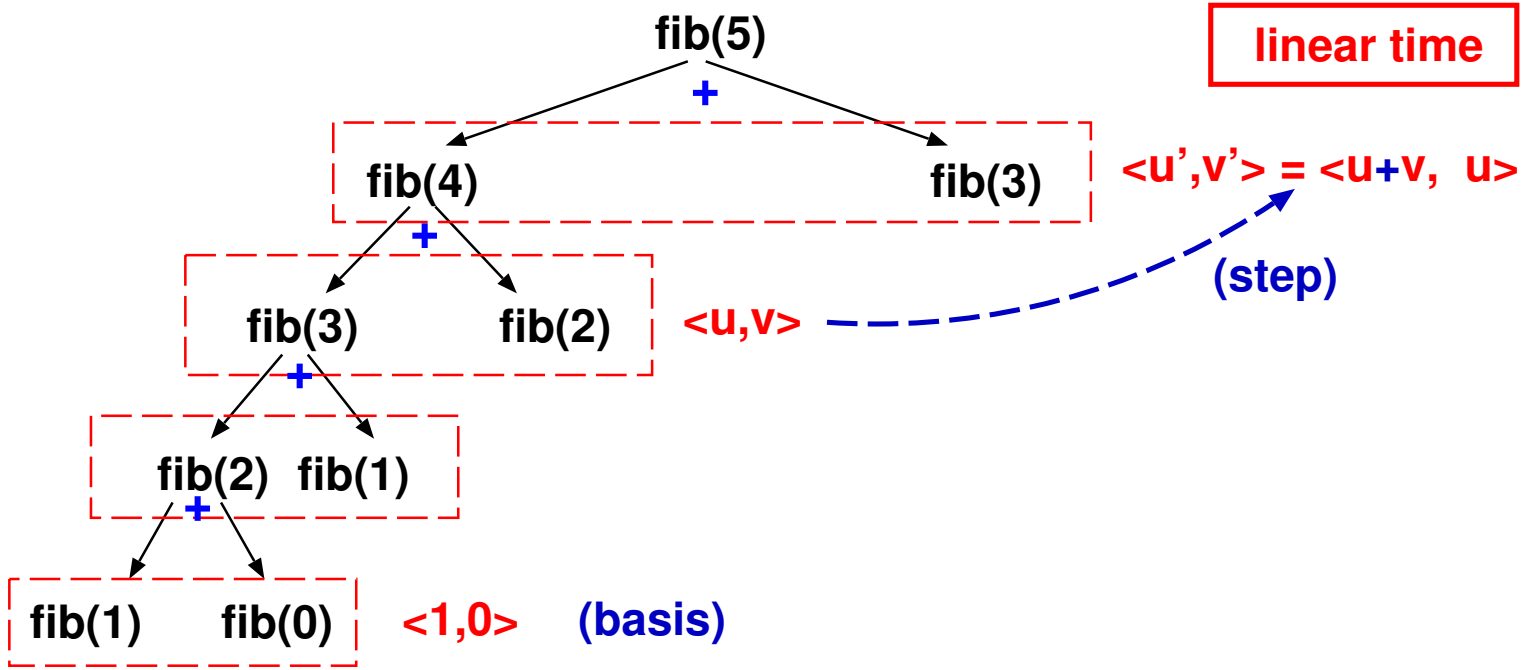
Fibonacci (1)

fib(0) = 0
fib(1) = 1
fib(n+2) = fib(n+1) + fib(n)



exponential time

#sums(n) = fib(n+1) -1



linear time

<u',v'> = <u+v, u>

Fibonacci (2)

1. fib(0) = 0
2. fib(1) = 1
3. fib(n+2) = fib(n+1) + fib(n)

↓ Unfolding 3 using 2 and 3

4.1 fib(2) = 1 + fib(0)

4.2 fib(n+3) = fib(n+1) + fib(n) + fib(n+1)

↓ Unfolding 4.1 using 1

5. fib(2) = 1

↓ where-abstraction of 4.2

6. fib(n+3) = 2 u + v where <u,v> = <fib(n+1), fib(n)>

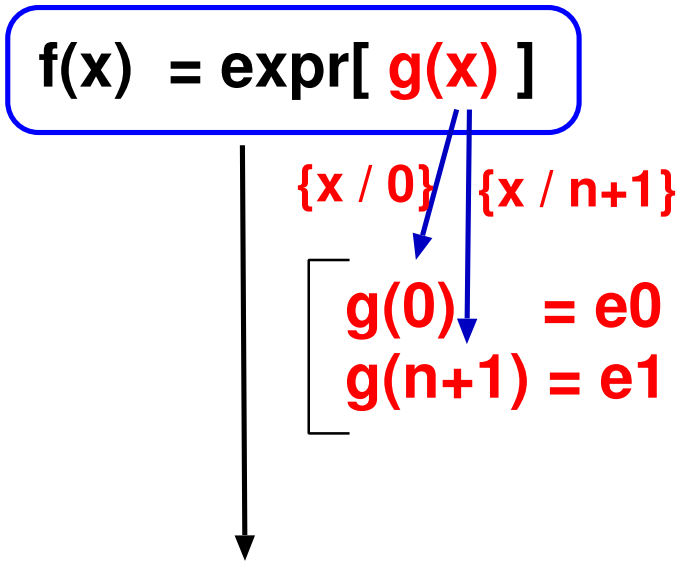
↓

...

R2. Unfolding

unfolding = instantiation ; replacement
{x / 0}{x / n+1} lhs → rhs

Given an equation:

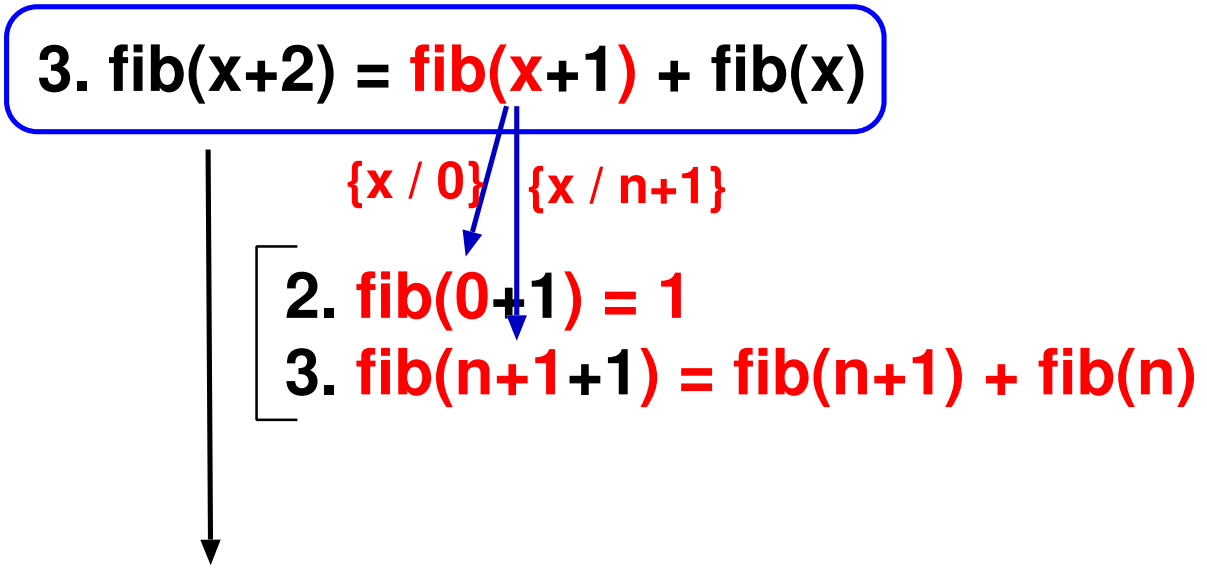


Replace it by the equations:

f(0) = expr {x / 0} [e0]
f(n+1) = expr {x / n+1} [e1]

R2. Unfolding fib(x+2)

Given an equation:



Replace it by the equations:

3. {x / 0}
3. {x / n+1}

4.1 fib(0+2) = 1 + fib(0)
4.2 fib(n+1+2) = fib(n+1) + fib(n) + fib(n+1)

Fibonacci (3)

...

↓

6. fib(n+3) = 2 u + v where <u,v> = <fib(n+1), fib(n)>

↓ Definition

7. g(n) = <fib(n+1), fib(n)>

↓ Folding 6 using 7

8. fib(n+3) = 2 u + v where <u,v> = g(n)

↓

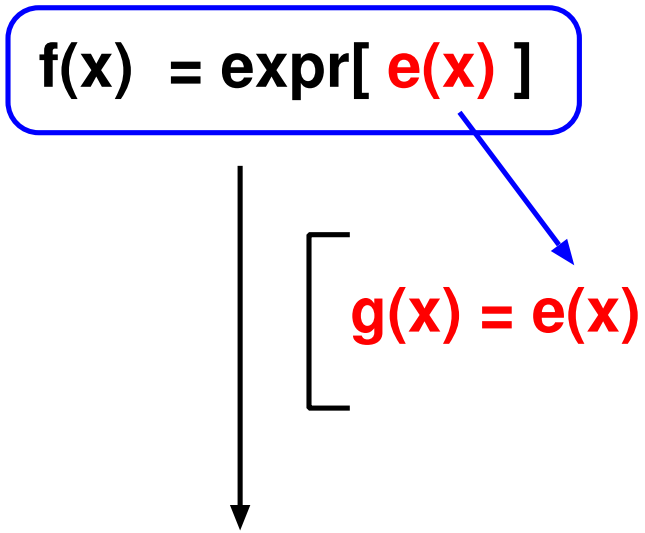
...

R3. Folding

folding = replacement

lhs ← rhs

Given an equation:



Replace it by the equation:

f(x) = expr [g(x)]

Fibonacci (4)

7. g(n) = <fib(n+1), fib(n)>

↓ Unfolding of 7 using 2 and 3

9.1 g(0) = <1, fib(0)>

9.2 g(n+1) = <fib(n+1)+fib(n), fib(n+1)>

↓ Unfolding of 9.1 using 1

10. g(0) = <1,0> (basis)

↓ where-abstraction of 9.2

11. g(n+1) = <u+v, u> where <u,v>=<fib(n+1), fib(n)>

↓ Folding of 11 using 7

12. g(n+1) = <u+v, u> where <u,v>=g(n) (step)

Fibonacci (5)

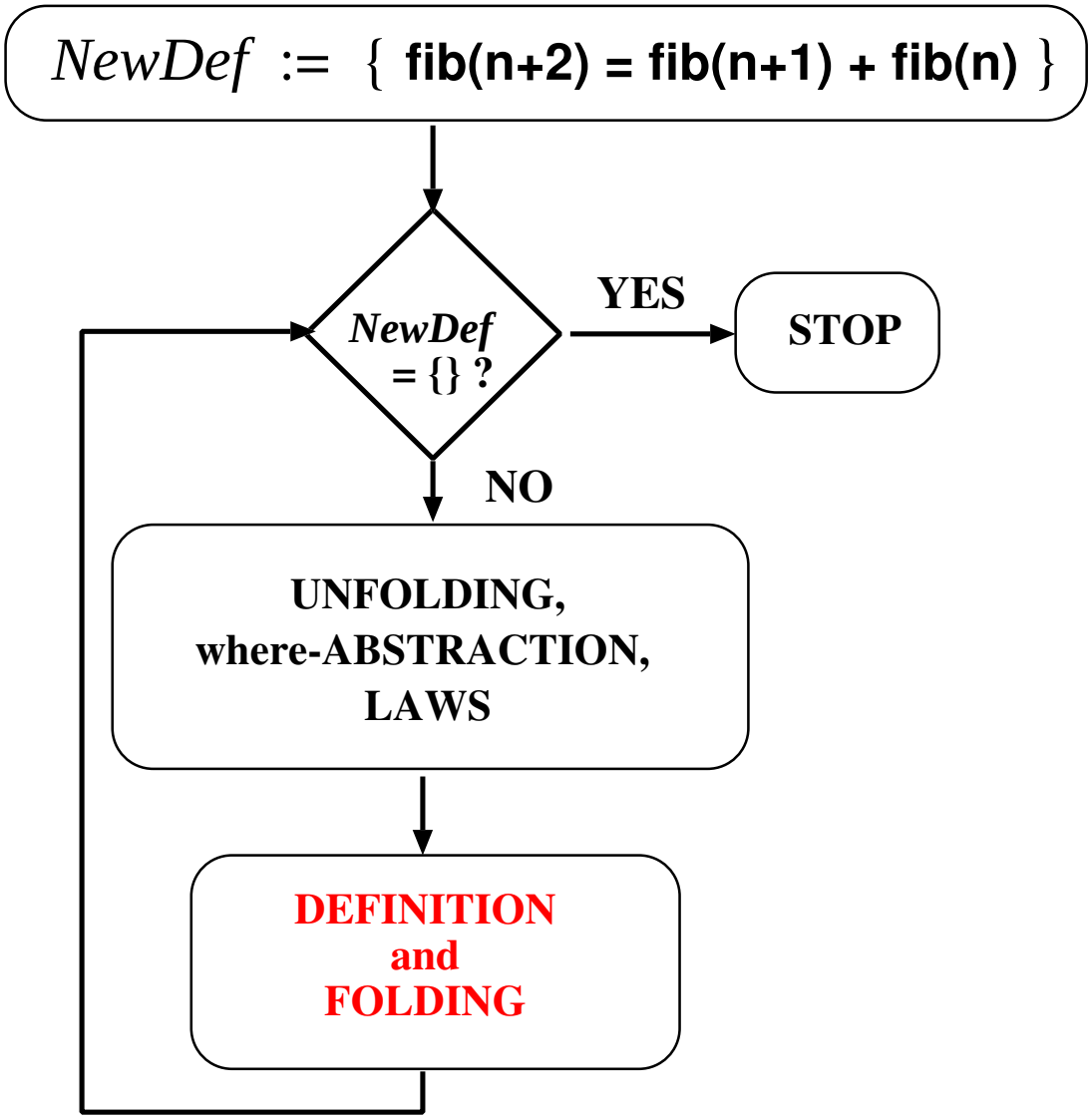
1. fib(0) = 0
2. fib(1) = 1
5. fib(2) = 1
8. fib(n+3) = 2u+v where <u,v> = g(n)
10. g(0) = <1,0> (basis)
12. g(n+1) = <u+v, u> where <u,v> = g(n) (step)

linear time

Transformation Rules

R1. Definition
R2. Unfolding
R3. Folding
R4. where-abstraction
R5. Laws (e.g., u+u = 2u)

The Unfold/Fold Transformation Strategy



Fibonacci (6)

Using the schema equivalence:

$g(0) = a$ (basis)

$g(n+1) = b(g(n))$ (step)

≡

```
{ n ≥ 0 }
G:=a;
while n>0 do n := n–1;
                G := b(G)
od
{ G = g(n) }
```

we get:

```
                                { n ≥ 0 }
if n=0 then F := 0 else
if n=1 then F := 1 else
if n=2 then F := 1 else
begin n := n-3;
        G := <1,0>;
        while n>0 do n := n–1;
                od  G := <G1+G2, G1>
        F := 2G1+G2
end                             { F = fib(n) }
```

linear time

- **Schema equivalences: correctness without Dijkstra's loop-invariants**

  The rules are shown to preserve semantics

  Partial correctness is preserved:

  1. f(0) = 1
  2. f(x) = f(x-1)

  ↓ Folding 2 using 2

  1. f(0) = 1
  3. f(x) = f(x)

  Termination is required to preserve total correctness

- **The "eureka problem": which definitions to introduce?**

  Like in Theorem Proving: which inductive hypothesis to use?

  A heuristics: tuple the calls that share common subcomputations
  (introduction of arrays)
  (introduction of lists)

Semantic limitations:

it does not exist a syntax-based set R of rules such that
given any two sets S1 and S2 of recursive equations which compute the same function,
we can transform S1 into S2 using R

Complexity limitations:

the depth of the call-tree cannot be decreased (without changing the algebra)

"algebra problem": which new algebra to introduce for improving efficiency?
By using multiplications we get a logarithmic Fibonacci computation

Computing Fibonacci (1)

using multiplications :

logarithmic time. 2 cells

$fib(n) = \pi_1\ r(n)$

$r(0) = \langle 0,1 \rangle$

$r(2k) = \langle u^2 + 2uv,\ u^2 + v^2 \rangle$ where $\langle u,v \rangle = r(k)$

$r(2k+1) = \langle (u+v)^2 + u^2,\ u^2 + 2uv \rangle$ where $\langle u,v \rangle = r(k)$

---

the first projection function π_1: $\pi_1 \langle a,b \rangle = a$

---

improving over matrix multiplication (Dijkstra et al.) :

logarithmic time. 4 cells

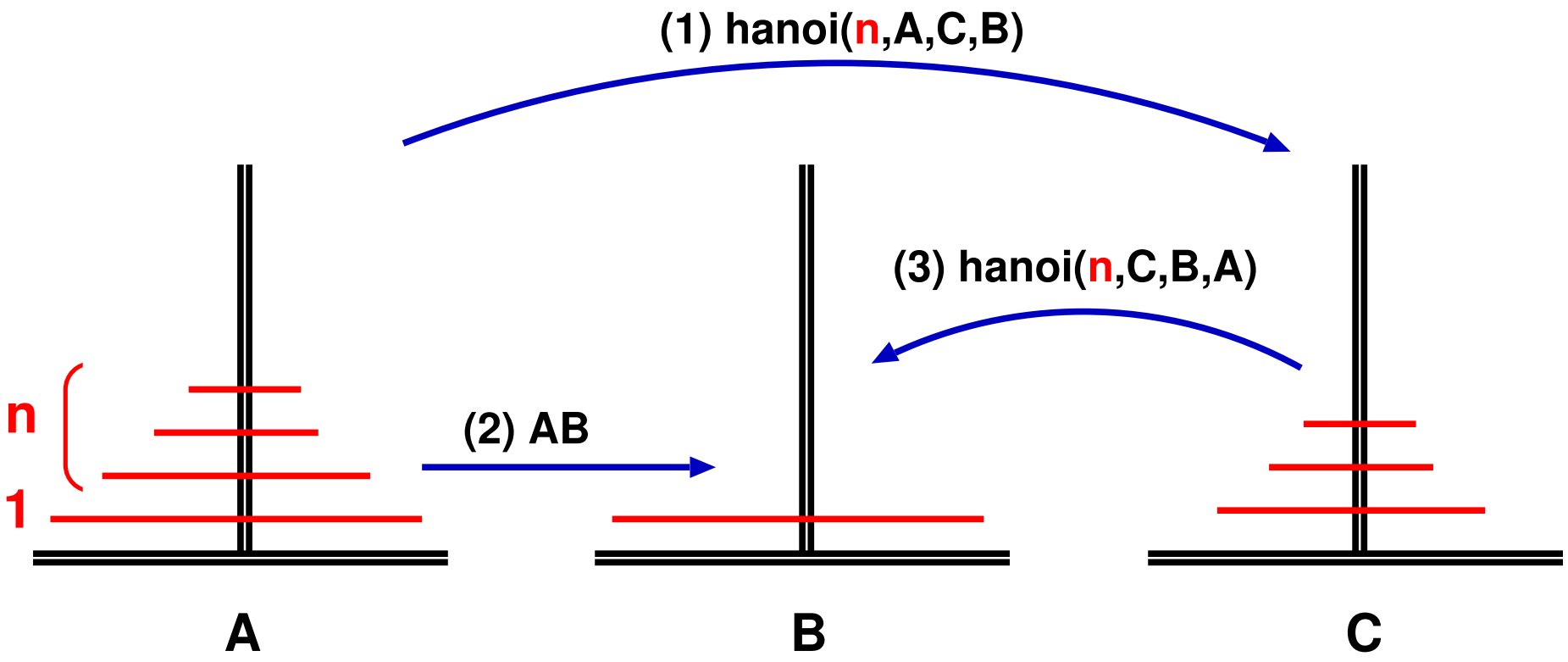
$$\begin{bmatrix} fib(n+1) \\ fib(n) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Given a matrix A, in order to compute A^n

- compute A^2, A^4, ... (no more than log n powers)
- add suitable powers of A (no more than log n additions)

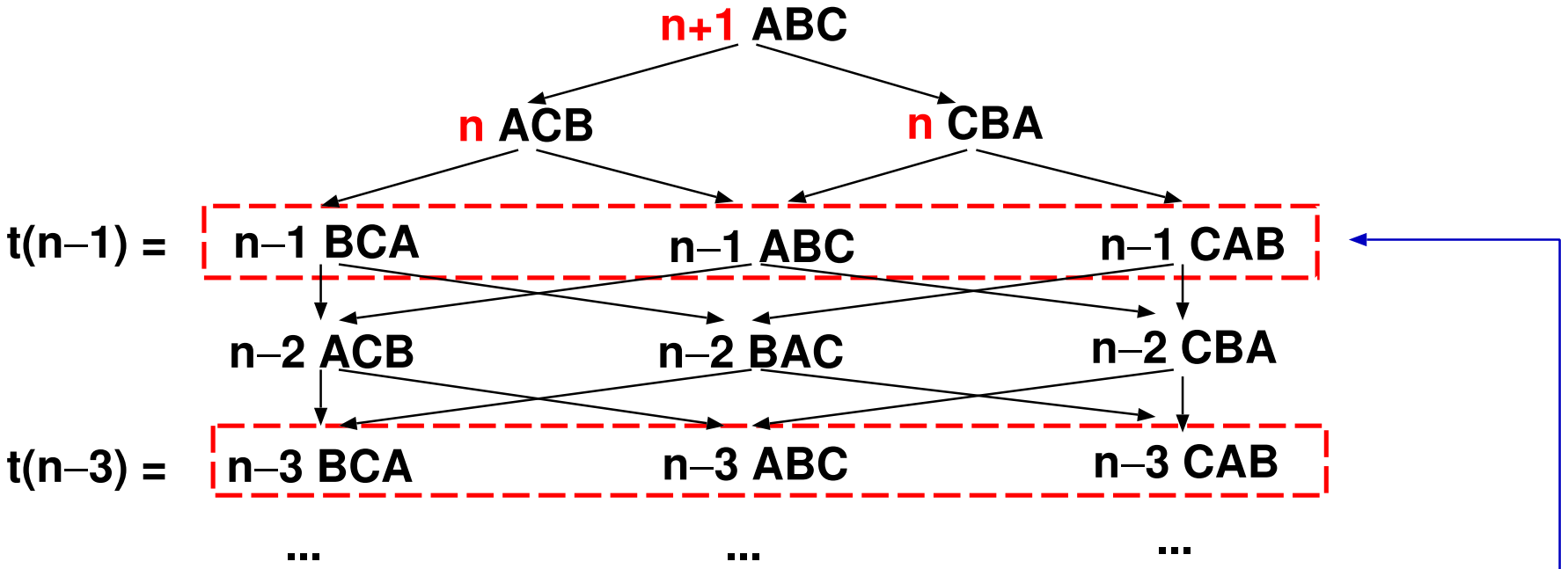
Towers of Hanoi (1)

hanoi(0,A,B,C) = skip
hanoi(n+1,A,B,C) = hanoi(n,A,C,B) : AB : hanoi(n,C,B,A)



Towers of Hanoi (2)

hanoi(0,A,B,C) = skip
hanoi(n+1,A,B,C) = hanoi(n,A,C,B) : AB : hanoi(n,C,B,A)



t(n) = < hanoi(n,B,C,A), hanoi(n,A,B,C), hanoi(n,C,A,B) >

---

t(0) = < skip, skip, skip >
t(1) = < BC, AB, CA >
t(n+2) = < u : BA : w : BC : v : AC : u,
v : AC : u : AB : w : CB : v,
w : CB : v : CA : u : BA : w > where <u,v,w> = t(n)

Towers of Hanoi (3)

Using the schema equivalence:

$h(0) = a$
$h(1) = b$
$h(n+2) = c(h(n))$

≡

```
{ N ≥ 0 }
n := N;
if even(n) then res := a else res := b;
while n>1 do res := c(res);
                n:= n–2
          od
    { res = h(N) }
```

we get:

```
                                    { N ≥ 0 }
var T, TT : array [1..3] of string;
var n, k : integer;
begin n := N;

    if  even(n)  then  for k := 1 to 3  do T[k] := "   "
                  else  begin T[1]:= "BC "; T[2]:= "AB "; T[3]:= "CA " end;
    while n>1 do
        begin for k := 1 to 3  do TT[k] := T[k];
            T[1] := TT[1] + "BA " + TT[3] + "BC " + TT[2] + "AC " + TT[1];
            T[2] := TT[2] + "AC " + TT[1] + "AB " + TT[3] + "CB " + TT[2];
            T[3] := TT[3] + "CB " + TT[2] + "CA " + TT[1] + "BA " + TT[3];
             n := n–2
        end

    res:=T[2];
end                                 { res = hanoi(N,A,B,C) }
```

Besides tupling, we can introduce new definitions by:

- **composition (avoids intermediate data structures)** $h(x) = f(g(x))$
- **generalization from expressions to variables**
- **lambda abstraction (higher order):**

```
palindrome(l) = eqlist(l, rev(l))
eqlist([],[]) = true
eqlist(a:l,b:m) = (a=b) and eqlist(l,m)
rev([]) = []
rev(a:l) = rev(l) :: [a]
```

↓

```
palindrome([]) = true
palindrome(a:l) = T(rev(a:l))  (a:l)
T([]) = λx. null(x)
T(a:l) = λx. a=hd(x)  and  T(l)  (tl(x))
```

a function

one traversal

(see Computing Surveys 1996, pp. 360-414)

Logic Program Transformation (Tamaki-Sato)

- A clause is a formula of the form: $H \leftarrow c \wedge L_1 \wedge \ldots \wedge L_n$

  where: – c is a constraint, i.e., a formula in a language of constraints,

  – each L_i is a literal, i.e., an atom A or a negated atom $\neg A$.

- A clause $p(\ldots) \leftarrow c \wedge L_1 \wedge \ldots \wedge L_n$ is stratified iff

  there exists a linear order $>$ on predicate symbols such that for i=1,...,n,

  – if L_i is the atom $q(\ldots)$ then $p \geq q$

  – if L_i is a negated atom $\neg q(\ldots)$ then $p > q$

  (no recursion through negation)

- A program P is a set of stratified clauses.

H. Tamaki, T. Sato: "Unfold/Fold Transformation of Logic Programs" In S.A. Tärlund, ed., Proceedings of the Second International Conference on Logic Programming, Uppsala, Sweden, 1984.

Logic Program Transformation (2)

- $p \leftarrow \neg q$ is stratified ($p > q$)

  $q \leftarrow q$

- $p \leftarrow \neg p$ is not stratified

Transformation Rules

R1. Definition
R2. Unfolding
R3. Folding
R4. Clause Removal
R5. Constraint Replacement

[Fioravanti-Pettorossi-Proietti 04]
extension of rules of : - Bensaou-Guessarian
- Etalle-Gabbrielli
- Gergatsoulis-Katzouraki
- Tamaki-Sato

R1. Definition

Introduce a definition:

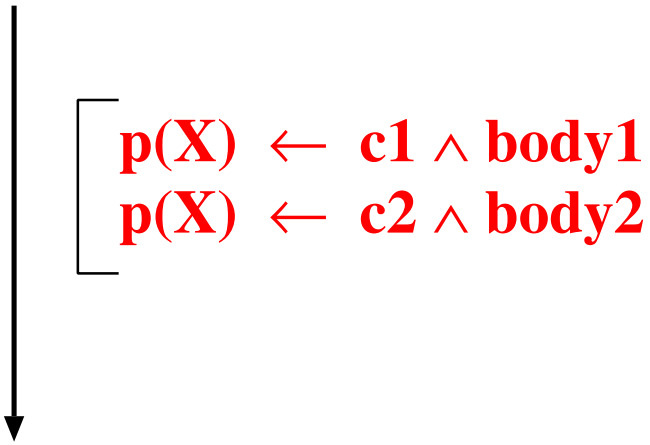
$$\text{newp}(X_1, ..., X_h) \leftarrow c \wedge G$$

where newp is a newp predicate symbol

R2. Positive Unfolding

Given a clause:

$$H \leftarrow c \wedge p(X) \wedge G$$



Replace it by the set of clauses:

$$H \leftarrow c \wedge c1 \wedge body1 \wedge G$$
$$H \leftarrow c \wedge c2 \wedge body2 \wedge G$$

and delete the clauses with unsatisfiable constraints

R2. Negative Unfolding (simplified)

Given a clause :

$$H \leftarrow c \wedge \neg p(t) \wedge G$$

$$p(X) \leftarrow d \wedge q(X)$$

$$\neg(d \wedge q(t))$$

$$(\neg d \vee (d \wedge \neg q(t))$$

Replace by:

$$H \leftarrow c \wedge \neg d \wedge G$$

$$H \leftarrow c \wedge d \wedge \neg q(t) \wedge G$$

R3. Folding (simplified)

Given a clause:

$$H \leftarrow c \wedge G1 \wedge B(X) \wedge G2$$

and a definition:

$$newp(X) \leftarrow d \wedge B(X)$$

such that $c \rightarrow d$

Replace by:

$$H \leftarrow c \wedge G1 \wedge newp(X) \wedge G2$$

R4. Clause Removal

- Remove clause with unsatisfiable constraints

$H \leftarrow c \wedge G$ — if c is unsatisfiable

- Remove Useless Clauses

$p \leftarrow q$

$q \leftarrow p$

$r \leftarrow$

no facts for p and q

- Remove Subsumed Clauses

$(H \leftarrow c \wedge G_1 \wedge G_2)\vartheta$

$H \leftarrow d \wedge G_2$

if $c \rightarrow d$

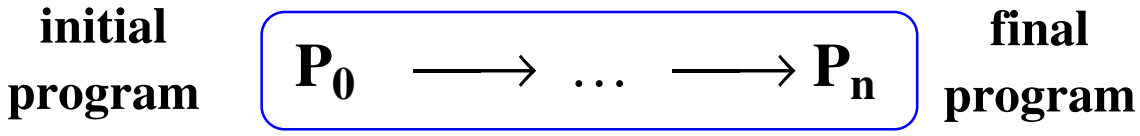
R5. Constraint Replacement

Given: $H \leftarrow c_1 \wedge G$

if $c_1 \leftrightarrow c_2$

Replace by: $H \leftarrow c_2 \wedge G$

Correctness of the Rules



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

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

Each rule application preserves the Perfect Model M.

Perfect Model (1)

- **Definite Programs (no negation, no constraints)**

P_0: $nat(0) \leftarrow$

$nat(s(X)) \leftarrow nat(X)$

$M(P_0) = \{nat(0), nat(s(0)), nat(s(s(0))), ...\}$ = least Herbrand model

- **Stratified Programs (negation, no constraints)** $(p > q)$

P_1: $p \leftarrow \neg q$

$q \leftarrow$

$\{\} \cup \{q\}$

$M(P_1) = \{q\}$

P_2: $p \leftarrow \neg q$

$q \leftarrow q$

$\{p\} \cup \{\}$

$M(P_2) = \{p\}$

stratum 2

stratum 1

Perfect Model (2)

- **Constraints** interpreted in an interpretation *D*.
- **Constraint Programs (no negation, constraints)** *D* is the Reals $\Re$

P_3: $p(X) \leftarrow X>Y \wedge q(Y)$

$q(X) \leftarrow X=0$

$M(P_3) = \{q(0)\} \cup \{p(X) \mid X \in \Re \wedge X>0\}$

- **Constraint Stratified Programs (negation, constraints)** (p > int)

D is the Reals $\Re$

P_4: $p(X) \leftarrow \neg\, int(X)$

$int(X) \leftarrow X=0$

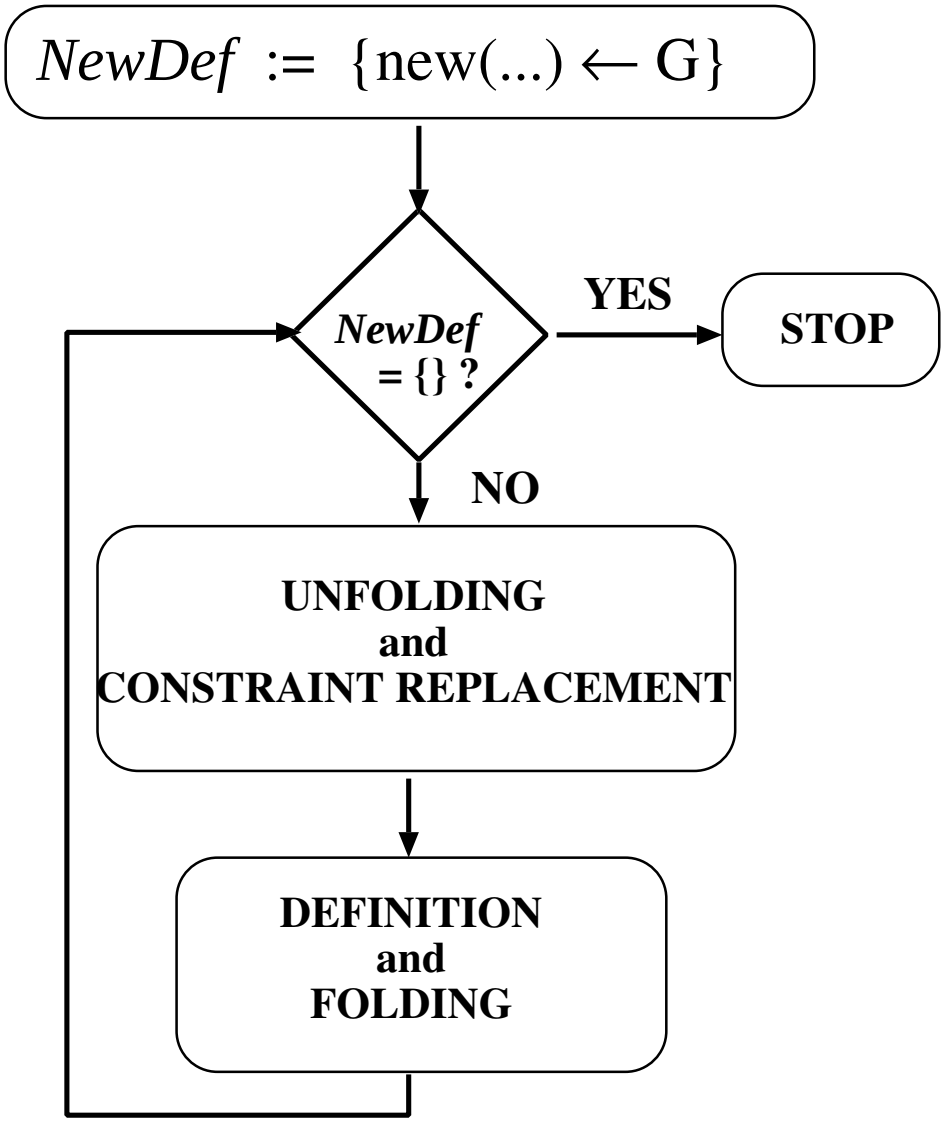
$int(X) \leftarrow X=Y+1 \wedge int(Y)$

$int(X) \leftarrow X=Y-1 \wedge int(Y)$

$M(P_4) = \{p(X) \mid X \in \Re \wedge X \text{ is not an integer}\} \cup$

$\{int(X) \mid X \in \Re \wedge X \text{ is an integer}\}$

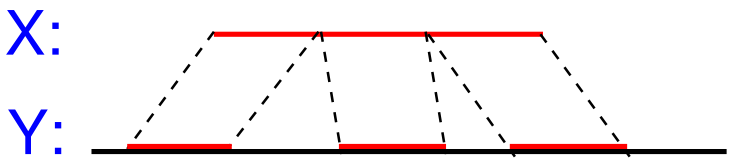
The Unfold/Fold Transformation Strategy



Common Subsequences (1)

- Given two sequences Y and Z, find a common subsequence X.

  subseq(X,Y) iff X is a subsequence of Y



  csub(X,Y,Z) iff X is a common subsequence of Y and Z

Initial Program:

1. csub(X,Y,Z) ← subseq(X,Y) ∧ subseq(X,Z)
2. subseq([ ],Y) ←
3. subseq([A|X],[A|Y]) ← subseq(X,Y)
4. subseq([A|X],[B|Y]) ← subseq([A|X],Y)

double visit to avoid

- csub(X,Y,Z) is solved by generating a subsequence X of Y and testing whether or not X is a subsequence of Z

Common Subsequences (2)

First Iteration

NewDef : 1. csub(X,Y,Z) ← subseq(X,Y) ∧ subseq(X,Z)

Unfolding

8. csub([ ],Y,Z) ←

9. csub([A|X],[A|Y],Z) ← subseq(X,Y) ∧ subseq([A|X],Z)

10. csub([A|X],[B|Y],Z) ← subseq([A|X],Y) ∧ subseq([A|X],Z)

Definition and Folding

add to *NewDef*

11. new1(A,X,Y,Z) ← subseq(X,Y) ∧ subseq([A|X],Z)

8. csub([ ],Y,Z) ←

9.f csub([A|X],[A|Y],Z) ← new1(A,X,Y,Z)

10.f csub([A|X],[B|Y],Z) ← csub([A|X],Y,Z)

add to final program:
no double occurrences
of variables

Common Subsequences (3)

Second Iteration

NewDef : 11. new1(A,X,Y,Z) ← subseq(X,Y) ∧ subseq([A|X],Z)

Unfolding

12. new1(A,X,Y,[A|Z]) ← subseq(X,Y) ∧ subseq(X,Z)
13. new1(A,X,Y,[B|Z]) ← subseq(X,Y) ∧ subseq([A|X],Z)

Folding (no new definition is needed)

12.f new1(A,X,Y,[A|Z]) ← csub(X,Y,Z)
13.f new1(A,X,Y,[B|Z]) ← new1(A,X,Y,Z)

add to final program:
no double occurrences
of variables

Common Subsequences (4)

Final Program:

8. csub([ ],Y,Z) ←
9.f csub([A|X],[A|Y],Z) ← new1(A,X,Y,Z)
10.f csub([A|X],[B|Y],Z) ← csub([A|X],Y,Z)
12.f new1(A,X,Y,[A|Z]) ← csub(X,Y,Z)
13.f new1(A,X,Y,[B|Z]) ← new1(A,X,Y,Z)

- No double occurrences of variables in the body
- The generation of a subsequence X of Y is interleaved with the test that X is a subsequence of Y.
  Thus, failure can occur before X is fully generated, and backtracking is reduced.

2. The Unfold/Fold Synthesis Method

The Unfold/Fold Synthesis Method

Given a program P and a first order formula $\varphi[X]$,
derive a program S defining a predicate s(X) such that

for all ground terms t, $M(P) \models \varphi[t]$ iff $M(S) \models s(t)$

- Step 1. Apply Lloyd-Topor transformation starting from:

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

and derive a stratified program Q such that

for all ground t, $M(P) \models \varphi[t]$ iff $M(P \cup Q) \models s(t)$

- Step 2. Apply the unfold/fold transformation strategy

$$P \cup Q \rightarrow \cdots \rightarrow S$$

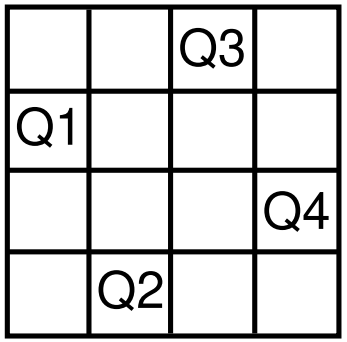
By the correctness of the transformation rules,

for all ground terms t, $M(P \cup Q) \models \varphi[t]$ iff $M(S) \models s(t)$

The N-Queens Problem

The N-Queens problem: Given an N×N board, place N queens such that any two queens do not attack each other.

A board configuration is a list L of N integers (i.e., rows).



column

row

rows: [2 , 4 , 1 , 3], i.e., queens: (1,2) (2,4) (3,1) (4,3)

The queen at (K,A) does not attack the queen at (M,B) iff

A≠B and |A-B|≠|M-K| (because K≠M)

different rows

different diagonals

Specification of N-Queens

The relation queens(N,L) is specified by the first order formula:

$$\varphi[N,L]: \quad \text{nat}(N) \wedge \text{natlist}(L) \wedge \text{length}(L,N) \wedge \quad (1)$$

$$\forall X\ (\text{member}(X,L) \rightarrow \text{in-range}(X,1,N)) \wedge \quad (2)$$

$$\forall A,B,K,M\ ((1 \leq K \wedge K \leq M \wedge \text{occurs}(A,K,L) \wedge \text{occurs}(B,M,L)) \quad (3)$$

$$\rightarrow (A \neq B \ \wedge \ A-B \neq M-K \ \wedge \ B-A \neq M-K))$$

meaning that:

- the list L is a permutation of the natural numbers between 1 and N (1)∧(2)
- for every K,M, with $1 \leq K \leq M$,

  if K is the A-th element of L and M is the B-th element of L, (3)

  then $A \neq B$ and $|A-B| \neq |M-K|$

different rows — different diagonals

... Specification of N-Queens

The relations nat(N), nat-list(L), length(L,N), member(X,L), in-range(X,M,N), occurs(X,I,L) are specified as follows:

P: nat(0) ←
nat(N) ← N=M+1 ∧ M≥0 ∧ nat(M)
nat-list([ ]) ←
nat-list([H|T]) ← nat(N) ∧ nat-list(T)
length([ ],0) ←
length([H|T]),N) ← N=M+1 ∧ M≥0 ∧ length(T,M)
member(X,[H|T]) ← X=H
member(X,[H|T]) ← member(X,[H|T])
in-range(X,M,N) ← X=N ∧ M≤N
in-range(X,M,N) ← N=K+1 ∧ M≤K ∧ in-range(X,M,K)
occurs(X,J,[H|T]) ← J=1 ∧ X=H
occurs(X,J+1,[H|T]) ← J≥1 ∧ occurs(X,J,T)

where: nat(N) iff N is a natural number
nat-list(L) iff L is a list of natural numbers
length(L,N) iff N is the length of list L
member(X,L) iff X is a member of list L
in-range(X,M,N) iff X is in [M,M+1, . . . , N]
occurs(X,J,L) iff X is the J-th element of list L

Synthesis of N-Queens

- Step 1. Apply Lloyd-Topor transformation starting from:

$$\text{queens}(N,L) \leftarrow \varphi[N,L]$$

and derive a stratified program Q such that

for all ground n and l $\quad M(P) \models \varphi[n,l]$ iff $M(P \cup Q) \models \text{queens}(n,l)$

- Step 2. Apply the unfold/fold transformation strategy

$$P \cup Q \rightarrow \cdots \rightarrow S$$

By the correctness of the transformation rules,

for all ground n and l $\quad M(P \cup Q) \models \varphi[n,l]$ iff $M(S) \models \text{queens}(n,l)$

Lloyd-Topor Transformation

Given a statement:

$A \leftarrow \varphi$

where A is an atom and φ is a first order formula,

we derive a stratified program.

$p(X) \leftarrow \forall Y\ (q(X,Y) \rightarrow r(Y))$

$\forall Y\ (\varphi 1 \rightarrow \varphi 2) \equiv \neg \exists Y (\varphi 1 \wedge \neg\ \varphi 2)$

$p(X) \leftarrow \neg\ \boxed{\exists Y\ (q(X,Y) \wedge \neg r(Y))}$

introduction of an auxiliary predicate

$p(X) \leftarrow \neg aux(X)$

$aux(X) \leftarrow \cancel{\exists Y}\ (q(X,Y) \wedge \neg r(Y))$

Y does not occur in aux(X)

$p(X) \leftarrow \neg aux(X)$

$aux(X) \leftarrow q(X,Y) \wedge \neg r(Y)$

Applying the LT-Transformation to N-Queens

Step 1. Derivation of a stratified program for queens(N,L)

$$queens(N,L) \leftarrow \varphi[N,L]$$

Lloyd-Topor Transformation

Q: $queens(N,L) \leftarrow nat(N) \wedge natlist(L) \wedge length(L,N) \wedge \neg aux1(L,N) \wedge \neg aux2(L)$

$aux1(L,N) \leftarrow member(X,L) \wedge \neg\, in\text{-}range(X,1,N)$

$aux2(L) \leftarrow \neg\, (1 \leq K \wedge K \leq M \wedge (A \neq B \wedge A-B \neq M-K \wedge B-A \neq M-K) \wedge occurs(A,K,L) \wedge occurs(B,M,L)$

Program $P \cup Q$ is very inefficient:

it generates a board configuration and tests that no queen is under attack

Deriving an Efficient Program for N-Queens (1)

Step 2. Derivation of a more efficient program for queens(N,L).
Repeatedly apply the unfold/fold transformation strategy, bottom-up on the strata of program Q.

Stratum 1: predicates aux1 and aux2.
Stratum 2: predicate queens.

Stratum 1

aux1(L,N) ← member(X,L) ∧ ¬ in-range(X,1,N) double occurrence, existential variable

unfolding ; folding

aux1([H|T],N) ← ¬ in-range(H,1,N)
aux1([H|T],N) ← aux1(T,N)

no double occurrences, no existential variables

Derivation of N-Queens (2)

. . . Stratum 1

$aux2(L) \leftarrow \neg\, (1 \leq K \wedge K \leq M \wedge (A \neq B \ \wedge\ A - B \neq M - K \ \wedge\ B - A \neq M - K) \wedge$
$occurs(A,K,L) \wedge occurs(B,M,L)$

Iteration 1:
unfolding ; constraint replacement ; definition ; folding

$aux2([A|T]) \leftarrow new1(A,T,0)$
$aux2([A|T]) \leftarrow aux2(T)$

$new1(A,L,J) \leftarrow 1 \leq M \wedge \neg\, (A \neq B \wedge A - B \neq M + J \wedge B - A \neq M + J) \wedge occurs(B,M,L)$

new definition

Iteration 2:
unfolding ; constraint replacement ; definition ; folding

$new1(A,[B|T],J) \leftarrow \neg\, (A \neq B \ \wedge\ A - B \neq J + 1 \ \wedge\ B - A \neq J + 1)$
$new1(A,[B|T],J) \leftarrow new1(A,T,J+1)$

Derivation of N-Queens (3)

Stratum 2

$$queens(N,L) \leftarrow nat(N) \wedge natlist(L) \wedge length(L,N) \wedge \neg aux1(L,N) \wedge \neg aux2(L)$$

(unfolding ; constraint replacement ; definition ; folding)^3

Final Program: S

$queens(N,L) \leftarrow new2(N,L,0)$

$new2(N,[\,],J) \leftarrow N=J$

$new2(N,[H|T\,],J) \leftarrow N \geq J+1 \wedge new2(N,T,J+1) \wedge new3(H,T,N,0)$

$new3(A,[\,],N,M) \leftarrow in\text{-}range(A,1,N) \wedge nat(A)$

$new3(A,[B|T\,],N,M) \leftarrow A \neq B \wedge A-B \neq M+1 \wedge B-A \neq M+1 \wedge nat(B) \wedge new3(A,T,N,M+1)$

S is a definite program.

Using the final program S, we detect that a queen is under attack before completing the board configuration.

3. The Unfold/Fold Verification Method

Program Verification

The Yale Shooting Problem

Program YSP :

holds(alive, []) ←
holds(loaded, [load|S]) ←
holds(dead, [shoot|S]) ← holds(loaded, S)
ab(alive, shoot, S) ← holds(loaded, S)
holds(F, [E|S]) ← fact(F) ∧ event(E) ∧ holds(F,S) ∧ ¬ab(F,E,S)
event(load) ←
event(shoot) ←
event(wait) ←
fact(alive) ←
fact(dead) ←
fact(loaded) ←
append([ ],Y,Y) ←
append([A|X],Y,[A|Z]) ← append(X,Y,Z)

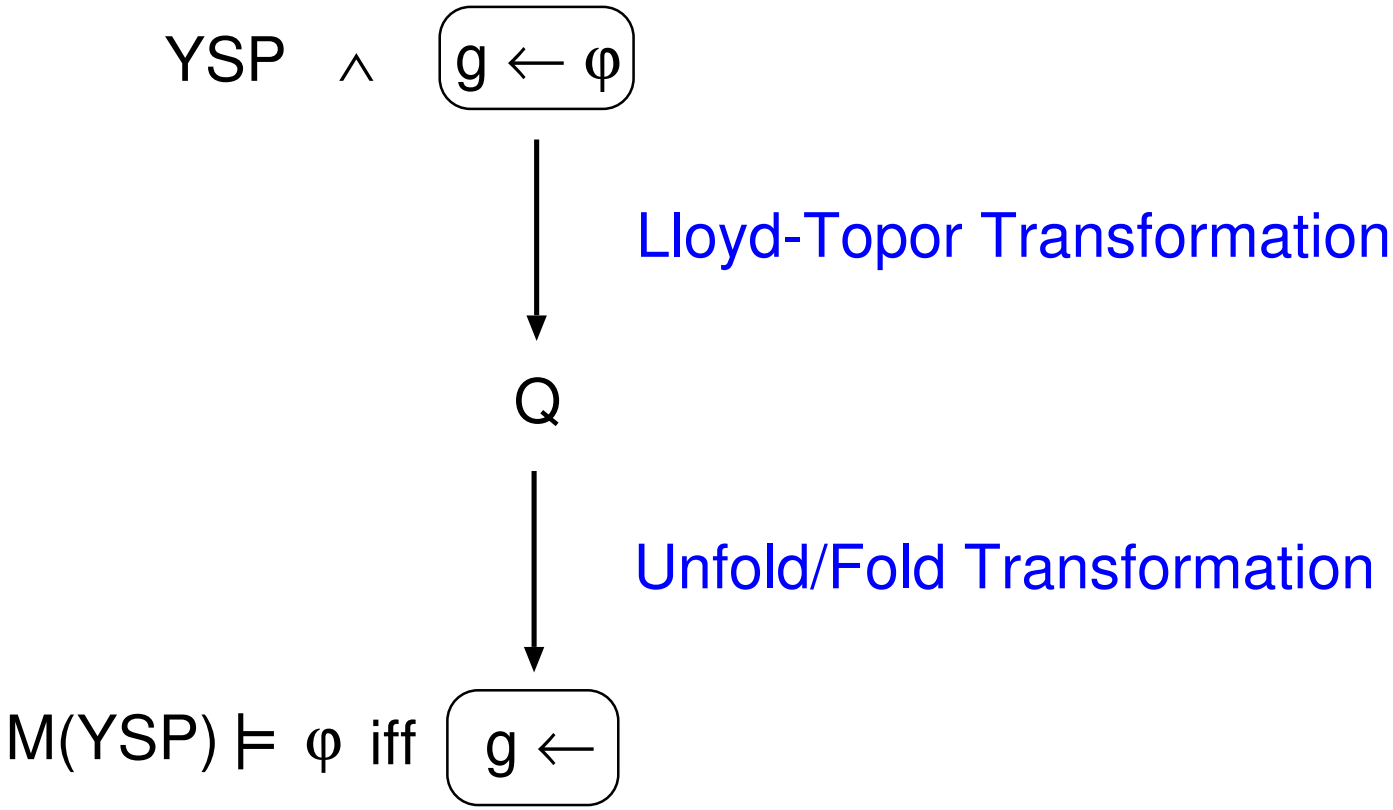
Property φ:

∀S (holds(dead,S) →
    ∃S1, S2, S3. (S = S1 shoot S2 loaded S3))

time flows from right to left

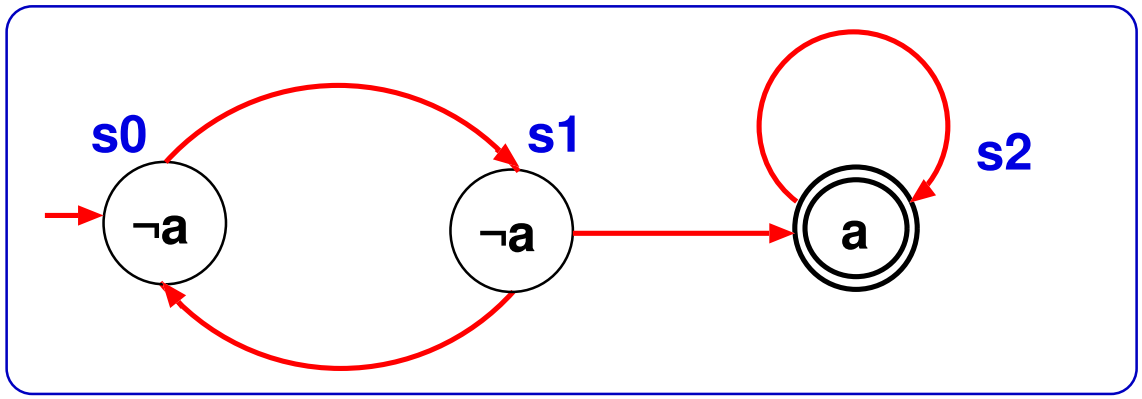
a sequence of events

Program Verification



Verifying a Finite-State System

K :



To prove: K,s0 ⊨ AG EF a

that is, K,s0 ⊨ ¬ EF ¬ EF a

we show: sat(s0, ¬ ef ¬ ef a) ∈ $M(P_K)$

P_K :

1. sat(s2, a) ←
2. sat(S, ¬ F) ← ¬ sat(S, F)
3. sat(S, ef F) ← sat(S, F)
4. sat(s0, ef F) ← sat(s1, ef F)
5. sat(s1, ef F) ← sat(s0, ef F)
6. sat(s1, ef F) ← sat(s2, ef F)
7. sat(s2, ef F) ← sat(s2, ef F)

new0 ← sat(s0, ¬ ef ¬ ef a)

Experimental Results using MAP

Protocol	Property	Time
Bakery (mutual exclusion)	**safety:** $\neg$**EF unsafe** **liveness:** **AG(wait_a** $\rightarrow$ **AF use_a)**	**0.2 s** **2.3 s**
Ticket (mutual exclusion)	**safety:** $\neg$**EF unsafe** **liveness:** **AG(wait_a** $\rightarrow$ **AF use_a)**	**0.6 s** **3.0 s**
Petri Net	$\neg$**EF(x=3)**	**0.1 s**
Berkeley RISC (cache coherence)	$\neg$**EF(dirty>1** $\wedge$ **shared>1)** $\neg$**EF(dirty>2)**	**1.3 s** **2.0 s**
Xerox Dragon (cache coherence	$\neg$**EF(dirty>1** $\wedge$ **shared_clean>1)** $\neg$**EF(dirty>1** $\wedge$ **shared_dirty>1)** $\neg$**EF(dirty>2)**	**0.9 s** **1.0 s** **1.3 s**
DEC Firefly (cache coherence)	$\neg$**EF(dirty>1** $\wedge$ **shared>1)** $\neg$**EF(dirty>2)**	**0.4 s** **0.4 s**
Illinois Univ. (cache coherence)	$\neg$**EF(dirty>1** $\wedge$ **shared>1)** $\neg$**EF(dirty>2)**	**0.3 s** **0.3 s**
MESI (cache coherence)	$\neg$**EF(dirty>1** $\wedge$ **shared>1)** $\neg$**EF(dirty>2)**	**0.2 s** **0.3 s**

More Transformation Strategies and Examples

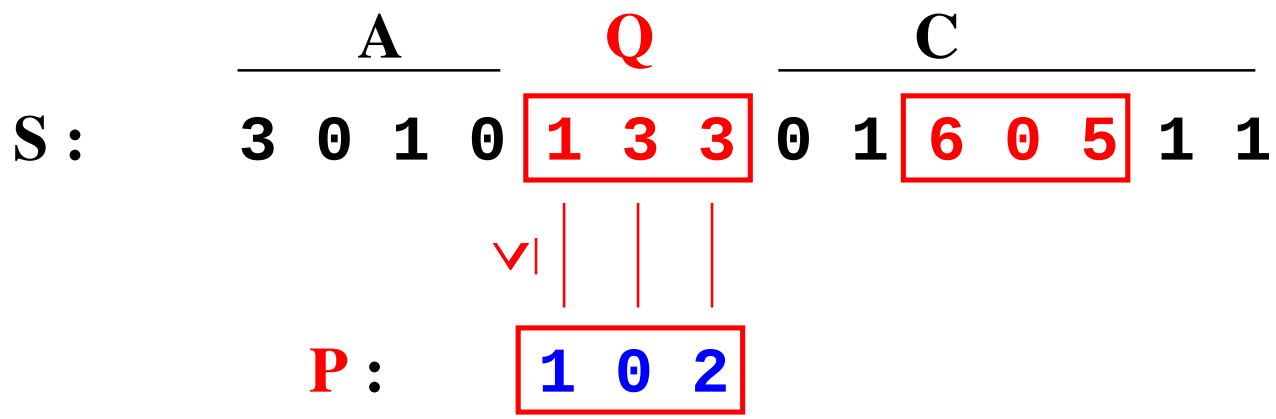
- **Specialization**: adapting programs to a context of use (parsers for cfl, string matching)
- **Determinization**: deriving deterministic programs from nondeterministic specifications
- **Verification** of Infinite State Systems (safety and liveness properties of protocols)

Leq_Match 1

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



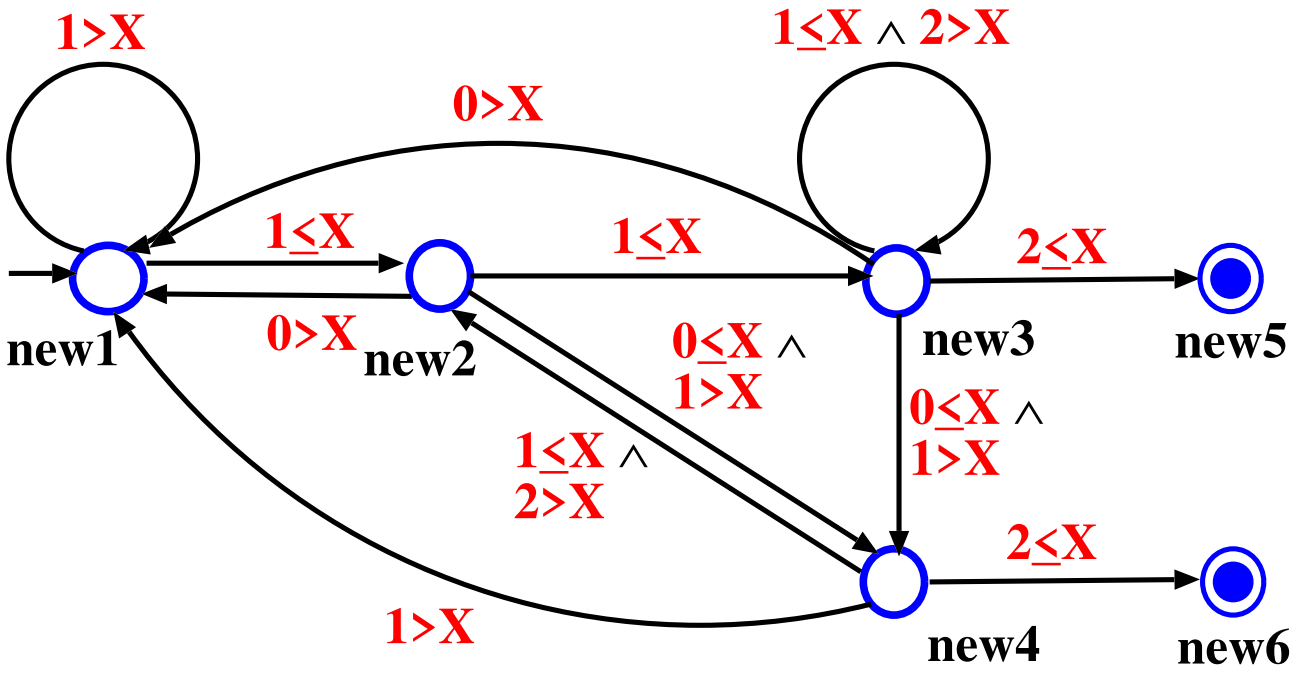
6. leq_match_{sp}(S) ← leq_match([1,0,2],S)

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. $new1([X|Xs]) \leftarrow 1 > X \wedge new1(Xs)$

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



mutual exclusive clauses → deterministic automaton

References

MAP Transformation System available at :
http://www.iasi.rm.cnr.it/~proietti/system.html

A. Pettorossi, M. Proietti: "Rules and Strategies for Transforming Functional and Logic Programs". ACM Computing Surveys, Vol. 28, No. 2, June 1996, pp. 360-414.

A. Pettorossi, M. Proietti, : "Program Derivation = Rules + Strategies". In: A. Kakas and F. Sadri (eds.) Computational Logic: Logic Programming and Beyond. Essays in honour of Bob Kowalski. Lecture Notes in Artificial Intelligence n. 2407, Springer-Verlag, 2002, pp. 273-309.

Fioravanti F., Pettorossi, A., Proietti, M.: "Automatic Proofs of Protocols via Program Transformations". Proc. of the Conference on "Monitoring, Security, and Rescue Techniques in Multiagent Systems'', Advances in Soft Computing (B. Dunin-Keplicz, A. Jankowski, A. Skowron, M. Szczuka, eds.) Springer, 2005. pp. 99-116.

Thanks to Jim, Ralf, and Andreas and each of you.