

# ***Program Transformation for Verification and Synthesis***

**Valerio Senni**

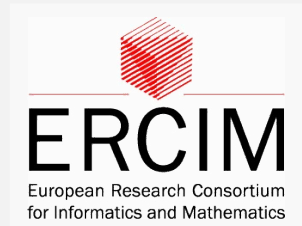
*Joint work with:*

*Fabio Fioravanti (Univ. D'Annunzio, Pescara, Italy),*

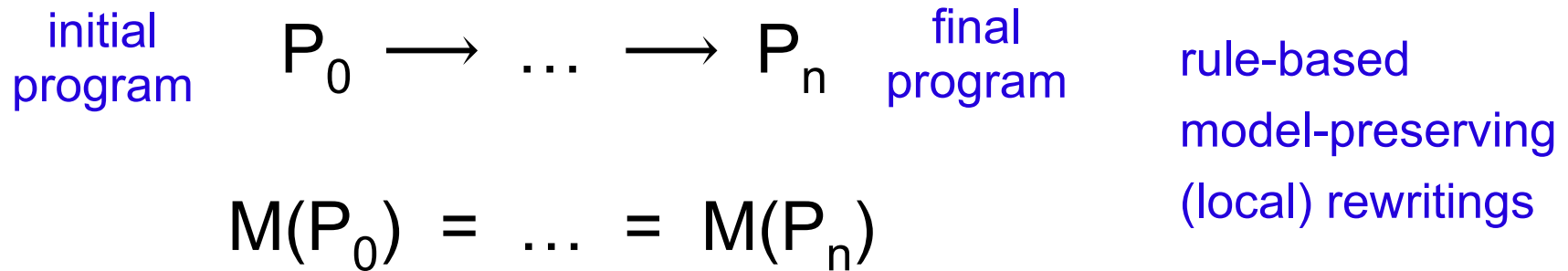
*Alberto Pettorossi (Univ. Tor Vergata, Rome, Italy),*

*Maurizio Proietti (IASI-CNR, Rome, Italy)*

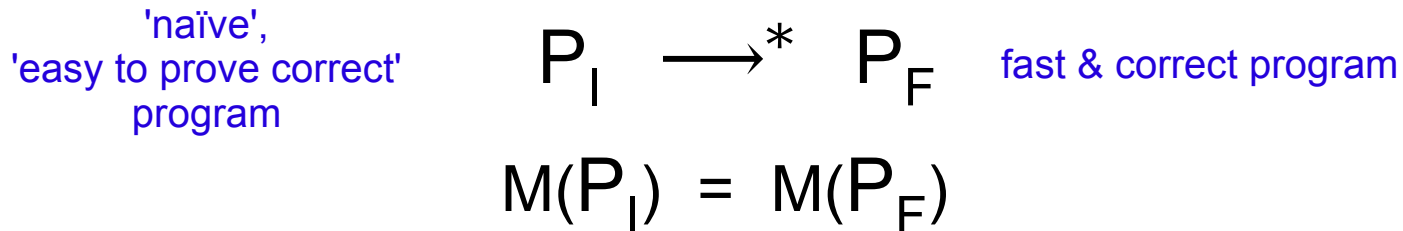
***EPFL, Lausanne, August 2010***



# Rule-Based Program Transformation



An approach to correct software development:



- Program optimization (for use into compilers)
- Program specialization (w.r.t. partially instantiated input)
- ...

# Transformation of Logic Programs

A program as a theory → theory transformation: changing the axioms of a theory while preserving its models

*Syntax:*

Logic programs

(i.e. sets of **horn clauses**

possibly with **negation** and / or **constraints**)

*Semantics:*

Least Herbrand Model

Perfect Model

Least *D*-Model

(depending on the use of negation and constraints)

Ideas from classical Program

Transformation can be used for:

- Theorem Proving
- Program Verification
- Program Synthesis

# Transformation Rules

$$\gamma: H \leftarrow G1 \wedge \mathbf{K\theta} \wedge G2$$

unfolding

$$\delta: \mathbf{K} \leftarrow \mathbf{B}$$

$\Rightarrow$

$$\eta: H \leftarrow G1 \wedge \mathbf{B\theta} \wedge G2$$

$$\eta: H \leftarrow G1 \wedge \mathbf{B\theta} \wedge G2$$

folding

$$\delta: \mathbf{K} \leftarrow \mathbf{B}$$

$\Rightarrow$

$$\gamma: H \leftarrow G1 \wedge \mathbf{K\theta} \wedge G2$$

$$M(P) \models \mathbf{G} \leftrightarrow \mathbf{G'}$$

goal  
replacement

$$\gamma: H \leftarrow G1 \wedge \mathbf{G} \wedge G2$$

$\Rightarrow$

$$\eta: H \leftarrow G1 \wedge \mathbf{G'} \wedge G2$$

Few, general rules

Dependent on the context

# Limitations of Program Transformation

There exists no complete set of transformation rules such that

If  $M(P_I) = M(P_F)$  then  $P_I \xrightarrow{R_1} \dots \xrightarrow{R_n} P_F$

even for Definite Logic Programs (that is, programs **without negation**)

# *Transformation of Logic Programs for Synthesis*

*'Push declarativeness to the limit'*

Logic Programming provides

an intuitive language for problem specification (Horn clauses) and  
a general reasoning method (Resolution)

It is very close (in syntax and semantics) to FOL

Program Transformation allows us to make LP even **more declarative**  
(FOL) and Resolution even more powerful (reasoning by Induction)

*From first order specifications to efficient and 'executable' programs*





# *An Introductory Example*

```
snm(S) ← near_match([2,0],2,S)
```

## *An Introductory Example*

$\text{snm}(S) \leftarrow \underline{\text{near\_match}([2,0],2,S)}$

Unfold  $\text{near\_match}([2,0],2,S)$  (a resolution step):

$\text{snm}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$

# An Introductory Example

$$\text{snm}(S) \leftarrow \text{near\_match}([2,0],2,S)$$

Unfold  $\text{near\_match}([2,0],2,S)$  (a resolution step):

$$\text{snm}(S) \leftarrow \underline{a(L,T,S)} \wedge \underline{a(Q,R,T)} \wedge n([2,0],2,Q)$$

Unfold\*

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 2 \wedge a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

recall :

$$a([],X,X) \leftarrow$$

$$a([X|Xs],Y,[X|Zs]) \leftarrow a(Xs,Y,Zs)$$

# An Introductory Example

$$\text{snm}(S) \leftarrow \text{near\_match}([2,0],2,S)$$

recall :

$$a([],X,X) \leftarrow$$

$$a([X|Xs],Y,[X|Zs]) \leftarrow a(Xs,Y,Zs)$$

Unfold  $\text{near\_match}([2,0],2,S)$  (a resolution step):

$$\text{snm}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

Unfold\*

1.  $\text{snm}([X|S]) \leftarrow 0 \leq X \leq 2 \wedge a(Q,R,S) \wedge n([0],2,Q)$

2.  $\text{snm}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$

$$\text{snm}([X|S]) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

By merging 1 and 2 and reasoning by cases we can determinize

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow X < 0 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow X > 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

mutually exclusive

# An Introductory Example

$$\text{snm}(S) \leftarrow \text{near\_match}([2,0],2,S)$$

Unfold  $\text{near\_match}([2,0],2,S)$  (a resolution step):

$$\text{snm}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

Fold (inverse of Unfold)

Unfold\*

1.  $\text{snm}([X|S]) \leftarrow 0 \leq X \leq 2 \wedge a(Q,R,S) \wedge n([0],2,Q)$

2.  $\text{snm}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$

$$\text{snm}([X|S]) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

By merging 1 and 2 and reasoning by cases we can determinize

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow X < 0 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow X > 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

mutually exclusive

tupling predicates  
that share variables

# An Introductory Example

$$\text{snm}(S) \leftarrow \text{near\_match}([2,0],2,S)$$

Unfold  $\text{near\_match}([2,0],2,S)$  (a resolution step):

$$\text{snm}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

Unfold\*

1.  $\text{snm}([X|S]) \leftarrow 0 \leq X \leq 2 \wedge a(Q,R,S) \wedge n([0],2,Q)$

2.  $\text{snm}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$

$$\text{snm}([X|S]) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

By merging 1 and 2 and reasoning by cases we can determinize

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow X < 0 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{snm}([X|S]) \leftarrow X > 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

**new1**  
Definition  
+ Fold

mutually exclusive

# An Introductory Example

By Folding, we get

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \text{new1}(S)$$

$$\text{snm}([X|S]) \leftarrow X < 0 \wedge \text{snm}(S)$$

$$\text{snm}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$$

where **new1** is defined as follows

$$\text{new1}(S) \leftarrow a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{new1}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

# An Introductory Example

By Folding, we get

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \text{new1}(S)$$

$$\text{snm}([X|S]) \leftarrow X < 0 \wedge \text{snm}(S)$$

$$\text{snm}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$$

where **new1** is defined as follows

$$\text{new1}(S) \leftarrow a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{new1}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

Unfold\* + case-split

$$\text{new1}([X|S]) \leftarrow -2 \leq X \leq 2$$

$$\text{new1}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{new1}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{new1}([X|S]) \leftarrow X < -2 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{new1}([X|S]) \leftarrow X > 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

mutually exclusive

# An Introductory Example

By Folding, we get

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \text{new1}(S)$$

$$\text{snm}([X|S]) \leftarrow X < 0 \wedge \text{snm}(S)$$

$$\text{snm}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$$

where **new1** is defined as follows

$$\text{new1}(S) \leftarrow a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{new1}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

Unfold\* + case-split

$$\text{new1}([X|S]) \leftarrow -2 \leq X \leq 2$$

$$\text{new1}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$$

$$\text{new1}([X|S]) \leftarrow 2 < X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$$

$$\text{new1}([X|S]) \leftarrow X < -2 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q) \rightarrow \text{snm}$$

$$\text{new1}([X|S]) \leftarrow X > 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q) \rightarrow \text{snm}$$

mutually exclusive

Fold

Fold

# An Introductory Example

The final program  $P_F$  :

$$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \text{new1}(S)$$
$$\text{snm}([X|S]) \leftarrow X < 0 \wedge \text{snm}(S)$$
$$\text{snm}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$$
$$\text{new1}([X|S]) \leftarrow -2 \leq X \leq 2$$
$$\text{new1}([X|S]) \leftarrow 2 < X \leq 4 \wedge \text{new1}(S)$$
$$\text{new1}([X|S]) \leftarrow X < -2 \wedge \text{snm}(S)$$
$$\text{new1}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$$

# An Introductory Example

The final program  $P_F$  :

$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \text{new1}(S)$

$\text{snm}([X|S]) \leftarrow X < 0 \wedge \text{snm}(S)$

$\text{snm}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$

$\text{new1}([X|S]) \leftarrow -2 \leq X \leq 2$

$\text{new1}([X|S]) \leftarrow 2 < X \leq 4 \wedge \text{new1}(S)$

$\text{new1}([X|S]) \leftarrow X < -2 \wedge \text{snm}(S)$

$\text{new1}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$

Correctness:

For all S

$M(P_I) \models \text{near\_match}([2,0],2,S)$

iff

$M(P_F) \models \text{snm}(S)$

where  $M( )$  denotes the least *D-model*

# An Introductory Example

The final program  $P_F$  :

$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \text{new1}(S)$

$\text{snm}([X|S]) \leftarrow X < 0 \wedge \text{snm}(S)$

$\text{snm}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$

$\text{new1}([X|S]) \leftarrow -2 \leq X \leq 2$

$\text{new1}([X|S]) \leftarrow 2 < X \leq 4 \wedge \text{new1}(S)$

$\text{new1}([X|S]) \leftarrow X < -2 \wedge \text{snm}(S)$

$\text{new1}([X|S]) \leftarrow X > 4 \wedge \text{snm}(S)$

Correctness:

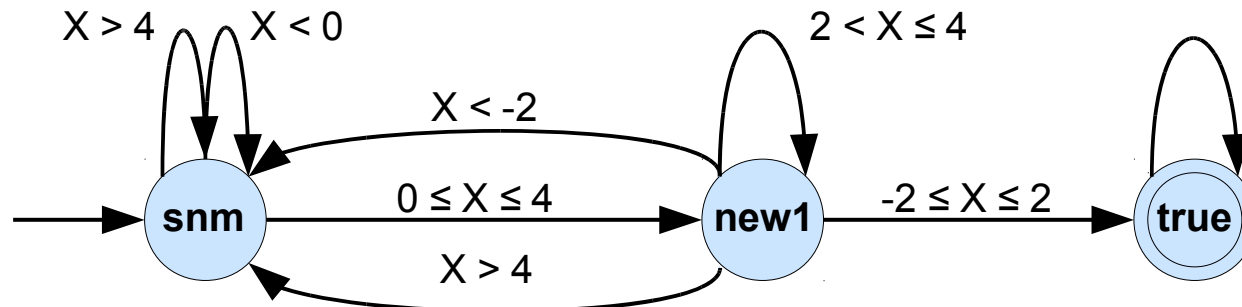
For all S

$M(P_I) \models \text{near\_match}([2,0],2,S)$

iff

$M(P_F) \models \text{snm}(S)$

where  $M(\ )$  denotes the least *D-model*



**Deterministic**

Example run using the MAP  
transformation system  
([www.iasi.cnr.it/~proietti/system.html](http://www.iasi.cnr.it/~proietti/system.html))



# Performed Optimizations

- Elimination of Existential Variables (nontermination on infinite domains)

$\text{near\_match}(P,K,S) \leftarrow \text{append}(L,T,S) \wedge \text{append}(Q,R,T) \wedge \text{near}(P,K,Q)$

- Elimination of Shared Variables (generate & test behaviour)

$\text{near\_match}(P,K,S) \leftarrow \text{append}(L,T,S) \wedge \text{append}(Q,R,T) \wedge \text{near}(P,K,Q)$

- Elimination of Nondeterminism (reduction of the search tree)

$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$

$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$



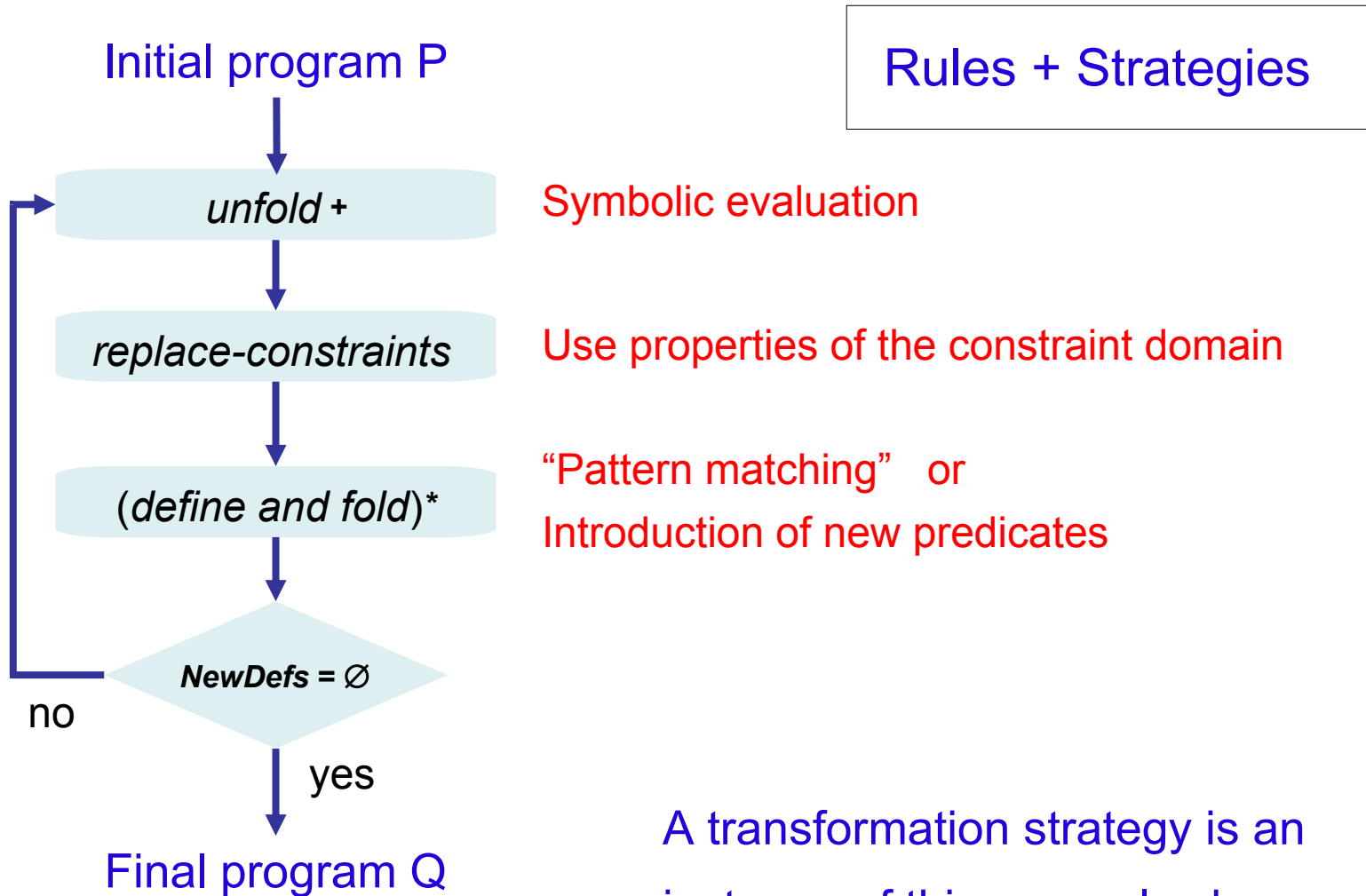
Folded into one clause

$\text{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \text{new1}(S)$

- Full Evaluation of Partial Input

$\text{snm}(S) \leftarrow \text{near\_match}([2,0],2,S)$

# General Unfold-Fold Transformation Scheme



A transformation strategy is an instance of this general scheme

## *Making LP 'More Declarative': an Example*

tree(e).

tree(t(L,R))  $\leftarrow$  tree(L)  $\wedge$  tree(R).

depth(e,0).

depth(t(L,R),D+1)  $\leftarrow$  depth(L,D).

depth(t(L,R),D+1)  $\leftarrow$  depth(R,D).

size(e,0).

size(t(L,R),SL+SR+1)  $\leftarrow$  size(L,SL)  $\wedge$  size(R,SR).

**balanced(T)  $\equiv$  tree(T)  $\wedge$   $\forall$  D1,D2 ( depth(T,D1)  $\wedge$  depth(T,D1)  $\rightarrow$  D1=D2 )**

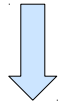
# Making LP 'More Declarative': an Example

tree(e).  
tree(t(L,R))  $\leftarrow$  tree(L)  $\wedge$  tree(R).

depth(e,0).  
depth(t(L,R),D+1)  $\leftarrow$  depth(L,D).  
depth(t(L,R),D+1)  $\leftarrow$  depth(R,D).

size(e,0).  
size(t(L,R),SL+SR+1)  $\leftarrow$  size(L,SL)  $\wedge$  size(R,SR).

balanced(T)  $\equiv$  tree(T)  $\wedge$   $\forall$  D1,D2 ( depth(T,D1)  $\wedge$  depth(T,D2)  $\rightarrow$  D1=D2 )



Lloyd-Topor transformation

balanced(T)  $\leftarrow$  tree(T)  $\wedge$   $\neg$  notbalanced(T).  
notbalanced(T)  $\leftarrow$  depth(T,**D1**)  $\wedge$  depth(T,**D2**)  $\wedge$   $\neg$  **D1=D2**.

balanced(T)

loops

size(T,4)  $\wedge$  balanced(T)

does not loop

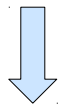
# Making LP 'More Declarative': an Example

tree(e).  
tree(t(L,R))  $\leftarrow$  tree(L)  $\wedge$  tree(R).

depth(e,0).  
depth(t(L,R),D+1)  $\leftarrow$  depth(L,D).  
depth(t(L,R),D+1)  $\leftarrow$  depth(R,D).

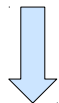
size(e,0).  
size(t(L,R),SL+SR+1)  $\leftarrow$  size(L,SL)  $\wedge$  size(R,SR).

balanced(T)  $\equiv$  tree(T)  $\wedge$   $\forall$  D1,D2 ( depth(T,D1)  $\wedge$  depth(T,D1)  $\rightarrow$  D1=D2 )



Lloyd-Topor transformation

balanced(T)  $\leftarrow$  tree(T)  $\wedge$   $\neg$  notbalanced(T).  
notbalanced(T)  $\leftarrow$  depth(T,**D1**)  $\wedge$  depth(T,**D2**)  $\wedge$   $\neg$  **D1=D2**.



Unfold/Fold transformation

balanced(e).  
balanced(t(e,e)).  
balanced(t(t(A,B),t(C,D)))  $\leftarrow$  balanced(t(A,C))  $\wedge$  balanced(t(B,C))  $\wedge$  balanced(t(A,D)).

balanced(T)

loops

size(T,4)  $\wedge$  balanced(T)

does not loop

balanced(T) does not loop

# Applications Roadmap

*From program  
optimization techniques*

*To 'reasoning' techniques*

Specialization

⇒

Infinite State Systems  
Model Checking

Elimination of Intermediate  
Data Structures

⇒

Proving Properties  
of Programs

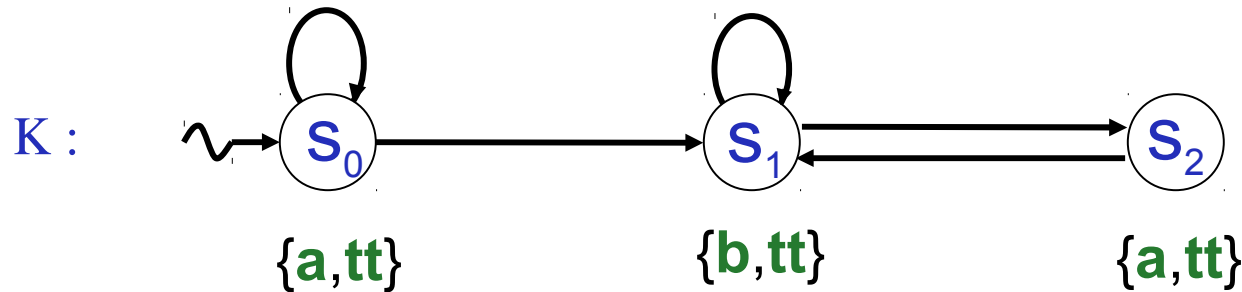
⇒

Synthesis

***Infinite State Systems***  
***Model Checking by***  
***Program Transformation***

# Model Checking Concurrent Systems

## Concurrent Systems as Kripke Structures



A computation path  $\pi = [s_0, s_1, \dots]$  is an infinite list of states of  $K$

## Properties as Formulas in CTL

$$\text{Elem} = \{a, b, tt\}$$

$$\varphi = \text{EU}(a, \text{AU}(tt, b))$$

## CTL Model Checking

Decide whether  $K, s \models \varphi$  or not.

# Semantics of CTL

Let  $K$  be a Kripke structure and  $s$  a state

Let  $\pi$  be an infinite list  $[s_0, s_1, \dots, s_k, \dots, s_n, \dots]$  of states

Let  $d, \varphi, \psi$  be formulas of CTL

$K, s \models d$  iff  $d \in \lambda(s)$

$K, s \models \neg \varphi$  iff  $K, s \models \varphi$  does not hold

$K, s \models \varphi \wedge \psi$  iff  $K, s \models \varphi$  and  $K, s \models \psi$

$K, s \models \mathbf{EX} \varphi$  iff  $\exists \pi = [s_0, s_1, \dots]$ ,  $s = s_0$ , and  $K, s_1 \models \varphi$

$K, s \models \mathbf{EU}(\varphi, \psi)$  iff  $\exists \pi = [s_0, s_1, \dots]$  s.t.  $s = s_0$  and  $\exists n \geq 0$   
 $((\forall k, 0 \leq k < n, K, s_k \models \varphi) \text{ and } K, s_n \models \psi)$

$K, s \models \mathbf{AF} \varphi$  iff  $\forall \pi = [s_0, s_1, \dots]$  if  $s = s_0$  then  $\exists n \geq 0$  s.t.  $K, s_n \models \varphi$

# Model Checking by Specialization

$\text{sat}(X, F) \leftarrow \text{elem}(X, F)$

$\text{sat}(X, \text{not}(F)) \leftarrow \neg \text{sat}(X, F)$

$\text{sat}(X, \text{and}(F1, F2)) \leftarrow \text{sat}(X, F1) \wedge \text{sat}(X, F2)$

$\text{sat}(X, \text{ex}(F)) \leftarrow \text{tr}(X, Y) \wedge \text{sat}(Y, F)$

$\text{sat}(X, \text{eu}(F1, F2)) \leftarrow \text{sat}(X, F2)$

Sat :  $\text{sat}(X, \text{eu}(F1, F2)) \leftarrow \text{sat}(X, F1) \wedge \text{tr}(X, Y) \wedge \text{sat}(Y, \text{eu}(F1, F2))$

$\text{sat}(X, \text{af}(F)) \leftarrow \text{sat}(X, F)$

$\text{sat}(X, \text{af}(F)) \leftarrow \text{ts}(X, Ys) \wedge \text{sat\_all}(Ys, \text{af}(F))$

$\text{sat\_all}([], F) \leftarrow$

$\text{sat\_all}([X|Xs], F) \leftarrow \text{sat}(X, F) \wedge \text{sat\_all}(Xs, F)$

Encoding the satisfiability relation of CTL

$K, s \models \varphi \text{ iff } M(\text{Sat}) \models \text{sat}(s, \varphi)$

# Model Checking by Specialization

$\text{sat}(X, F) \leftarrow \text{elem}(X, F)$

$\text{sat}(X, \text{not}(F)) \leftarrow \neg \text{sat}(X, F)$

$\text{sat}(X, \text{and}(F1, F2)) \leftarrow \text{sat}(X, F1) \wedge \text{sat}(X, F2)$

$\text{sat}(X, \text{ex}(F)) \leftarrow \text{tr}(X, Y) \wedge \text{sat}(Y, F)$

$\text{sat}(X, \text{eu}(F1, F2)) \leftarrow \text{sat}(X, F2)$

Sat :  $\text{sat}(X, \text{eu}(F1, F2)) \leftarrow \text{sat}(X, F1) \wedge \text{tr}(X, Y) \wedge \text{sat}(Y, \text{eu}(F1, F2))$

$\text{sat}(X, \text{af}(F)) \leftarrow \text{sat}(X, F)$

$\text{sat}(X, \text{af}(F)) \leftarrow \text{ts}(X, Ys) \wedge \text{sat\_all}(Ys, \text{af}(F))$

$\text{sat\_all}([], F) \leftarrow$

$\text{sat\_all}([X|Xs], F) \leftarrow \text{sat}(X, F) \wedge \text{sat\_all}(Xs, F)$

Encoding the satisfiability relation of CTL

$K, s \models \varphi \text{ iff } M(\text{Sat}) \models \text{sat}(s, \varphi)$

Example-specific data

$\text{prop} \leftarrow \text{sat}(s_0, \varphi)$

$\text{tr}(S_1, S_2) \leftarrow$

$\text{elem}(S, E) \leftarrow$

...

...

# Model Checking by Specialization

sat(X, F) ← elem(X,F)

sat(X, not(F)) ← ¬ sat(X,F)

sat(X, and(F1,F2)) ← sat(X,F1) ∧ sat(X,F2)

sat(X, ex(F)) ← tr(X,Y) ∧ sat(Y,F)

sat(X, eu(F1,F2)) ← sat(X,F2)

Sat : sat(X, eu(F1,F2)) ← sat(X,F1) ∧ tr(X,Y) ∧ sat(Y,eu(F1,F2))

sat(X, af(F)) ← sat(X,F)

sat(X, af(F)) ← ts(X,Ys) ∧ sat\_all(Ys,af(F))

sat\_all([ ],F) ←

sat\_all([X|Xs],F) ← sat(X,F) ∧ sat\_all(Xs,F)

Encoding the satisfiability relation of CTL

$K,s \models \varphi$  iff  $M(\text{Sat}) \models \text{sat}(s,\varphi)$

Example-specific data

prop ← sat(s<sub>0</sub>, φ)

tr(S<sub>1</sub>,S<sub>2</sub>) ←

elem(S,E) ←

...

...

Specialize !



Output is propositional

# Model Checking Infinite State Systems

By adding **Constraints**, we can do Infinite State Systems Model Checking

$\langle 3, 7 \rangle$  States as a tuples of elements of a constraint domain

$\langle X, Y \rangle \in X > Y$  Sets of states,

$\mathbf{tr}(\langle X, Y \rangle, \langle X', Y' \rangle) \leftarrow X' = Y + X \wedge Y' = Y - 1$  transitions, and

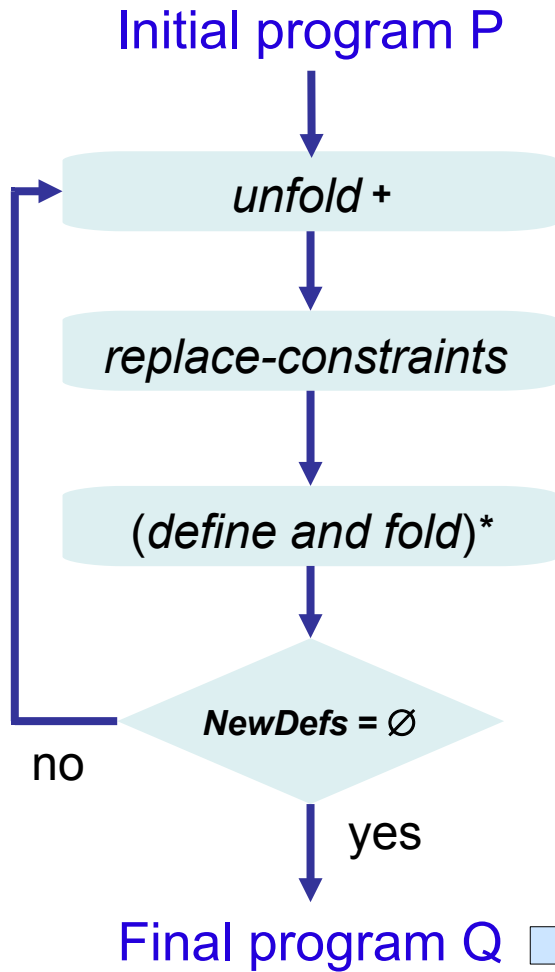
$\mathbf{initial}(\langle X, Y \rangle) \leftarrow X > 0 \wedge Y + X < 5$  elementary properties  
as constraints

$\text{prop} \leftarrow \neg \text{negprop}$

$\text{negprop} \leftarrow \text{initial}(S) \wedge \neg \text{sat}(S, \varphi) = \forall S \text{ initial}(S) \rightarrow \text{sat}(S, \varphi)$

Use of **generalization** techniques to make the transformation strategy terminate

# The Role of Generalization: Termination



**Constraint Generalization Techniques**  
(anticipate future needs for folding)

**Not propositional** (due to generalization)  
Requires more powerful analysis

# Comparison with other MC Tools

EXAMPLE	MAP	ALV				DMC		HyTech	
	<i>SC&amp;WidenPlus</i>	<i>default</i>	<i>A</i>	<i>F</i>	<i>L</i>	<i>noAbs</i>	<i>Abs</i>	<i>Fw</i>	<i>Bw</i>
Bakery 2 (safety)	20	20	30	90	30	10	30	$\infty$	20
Bakery 2 (liveness)	70	30	30	90	30	60	70	$\times$	$\times$
Bakery 3 (safety)	160	580	570	$\infty$	600	460	3090	$\infty$	360
MutAst	140	$\perp$	$\perp$	910	$\perp$	150	1370	70	130
Peterson N	230	71690	$\perp$	$\infty$	$\infty$	$\infty$	$\infty$	70	$\infty$
Ticket (safety)	40	$\infty$	80	30	$\infty$	$\infty$	60	$\infty$	$\infty$
Ticket (liveness)	110	$\infty$	230	40	$\infty$	$\infty$	220	$\times$	$\times$
Berkeley RISC	30	10	$\perp$	20	60	30	30	$\infty$	20
DEC Firefly	20	10	$\perp$	20	80	50	80	$\infty$	20
IEEE Futurebus+	2460	320	$\perp$	$\infty$	670	4670	9890	$\infty$	380
Illinois University	20	10	$\perp$	$\infty$	140	70	110	$\infty$	20
MESI	30	10	$\perp$	20	60	40	60	$\infty$	20
MOESI	60	10	$\perp$	40	100	50	90	$\infty$	10
Synapse N+1	10	10	$\perp$	10	30	0	0	$\infty$	0
Xerox PARC Dragon	40	20	$\perp$	40	340	70	120	$\infty$	20
Barber	1170	340	$\perp$	90	360	140	230	$\infty$	90
Bounded Buffer	3540	0	10	$\infty$	20	20	30	$\infty$	10
Unbonded Buffer	3890	10	10	40	40	$\infty$	$\infty$	$\infty$	20
CSM	6580	79490	$\perp$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Insertion Sort	100	40	60	$\infty$	70	30	80	$\infty$	10
Selection Sort	190	$\infty$	390	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Office Light Control	50	20	20	30	20	10	10	$\infty$	$\infty$
Reset Petri Nets	0	$\infty$	$\perp$	$\infty$	10	0	0	$\infty$	10

- Compared several generalization techniques
- Two-phase analysis

[FPPS10] (LOPSTR)

Experiments run using the MAP transformation system

# Related and Future Work

More expressive logics for path properties:

- LTL / CTL\* [PPS09] (LOPSTR)
- $\omega$ -regular languages [PPS10] (ICLP)

Future Work

- proving properties of logic programs on infinite structures (some work already in [PPS10] for infinite lists)
- synthesis of reactive systems

# *Proving Program Properties by Program Transformation*

# Main Idea

$$\text{match}(P,S) \leftarrow a(L,\mathbf{T},S) \wedge a(P,R,\mathbf{T})$$

- $\mathbf{T}$  is 'unnecessary' and its construction can be avoided
- $\mathbf{T}$  is 'existential' [recall:  $\forall X (p \leftarrow q(X)) \equiv p \leftarrow \exists X (q(X))$  ]
- (By transformation) We can find a better program:

$$\begin{aligned}\text{match}(P,S) &\leftarrow \text{prefix}(P,S) \\ \text{match}(P,[X|S]) &\leftarrow \text{match}(P,S)\end{aligned}$$

$$\begin{aligned}\text{prefix}([ ],S) &\leftarrow \text{true} \\ \text{prefix}([X|P],[X|S]) &\leftarrow \text{prefix}(P,S)\end{aligned}$$

# Main Idea

$$\text{match}(P,S) \leftarrow a(L,\mathbf{T},S) \wedge a(P,R,\mathbf{T})$$

- **T** is 'unnecessary' and its construction can be avoided
- **T** is 'existential' [recall:  $\forall X (p \leftarrow q(X)) \equiv p \leftarrow \exists X (q(X))$  ]
- (By transformation) We can find a better program:

$$\text{match}(P,S) \leftarrow \text{prefix}(P,S)$$

$$\text{match}(P,[X|S]) \leftarrow \text{match}(P,S)$$

$$\text{prefix}([ ],S) \leftarrow$$

$$\text{prefix}([X|P],[X|S]) \leftarrow \text{prefix}(P,S)$$

Idea: **Quantifiers Elimination** as 'Existential' Variables Elimination

# The Transformational Proof Method

We are given a CLP( $R_{lin}$ ) program

P:  $\text{member}(X,[Y|L]) \leftarrow X=Y$   
 $\text{member}(X,[Y|L]) \leftarrow \text{member}(X,L)$

and a closed first order formula

$\varphi: \forall L \exists U \forall X ( \text{member}(X,L) \rightarrow X \leq U )$

we want to prove:

$M(P) \models \varphi$

## LR-Programs

Namely: - linear recursion  
- no existential variables  
(this has a role in Unfolding+)

*Turing complete*

# Step 1. Clause-Form Transformation

$$\varphi: \quad \forall L \exists U \forall X ( \text{member}(X,L) \rightarrow X \leq U )$$

$$\text{prop} \leftarrow \neg \exists L \neg \exists U \neg \exists X ( \text{member}(X,L) \wedge \neg X \leq U )$$

By a variant of Lloyd-Topor transformation, we get a set CF of clauses (Clause-Form)

- CF: D4:  $\text{prop} \leftarrow \neg p$
- D3:  $p \leftarrow \text{list}(L) \wedge \neg q(L)$
- D2:  $q(L) \leftarrow \text{list}(L) \wedge \neg r(L,U)$
- D1:  $r(L,U) \leftarrow X > U \wedge \text{list}(L) \wedge \text{member}(X,L)$
- stratified program
  - **not** LR-clauses (e.g. with **existential variables**)

Such that  $M(P) \models \varphi$  iff  $M(P \cup CF) \models \text{prop}$

## Step 2. Bottom-Up Elimination of Existential Vars

By *unfold/fold* transformation from  $P \cup CF$  we derive a *propositional* program

Prop:  $\text{prop} \leftarrow \neg p$

$p \leftarrow p1$

$p1 \leftarrow p1$

s.t.  $M(P \cup CF) \models \text{prop}$  iff  $M(\text{Prop}) \models \text{prop}$

Thus,  $M(P) \models \varphi$  iff  $M(\text{Prop}) \models \text{prop}$

The transformation from  $P \cup CF$  to Prop consists in  
*eliminating the existential variables* from CF

i.e., the variables occurring in the body of a clause and not in the head

# *Eliminating Existential Variables via U/F*

D1:  $r(L,U) \leftarrow \exists X > U \wedge \text{list}(L) \wedge \text{member}(X,L)$

# Eliminating Existential Variables via U/F

D1:  $r(L,U) \leftarrow X > U \wedge \underline{\text{list}(L)} \wedge \underline{\text{member}(X,L)}$

Unfold:  $r([X|T],U) \leftarrow X > U \wedge \text{list}(T)$

$r([X|T],U) \leftarrow Y > U \wedge \text{list}(T) \wedge \text{member}(Y,T)$

# Eliminating Existential Variables via U/F

D1:  $r(L,U) \leftarrow \boxed{X > U \wedge \text{list}(L) \wedge \text{member}(X,L)}$

Unfold:  $r([X|T],U) \leftarrow X > U \wedge \text{list}(T)$

$r([X|T],U) \leftarrow \boxed{Y > U \wedge \text{list}(T) \wedge \text{member}(Y,T)}$



# Eliminating Existential Variables via U/F

D1:  $r(L,U) \leftarrow X > U \wedge \text{list}(L) \wedge \text{member}(X,L)$

Unfold:  $r([X|T],U) \leftarrow X > U \wedge \text{list}(T)$

$r([X|T],U) \leftarrow Y > U \wedge \text{list}(T) \wedge \text{member}(Y,T)$

Fold:  $r([X|T],U) \leftarrow X > U \wedge \text{list}(T)$

$r([X|T],U) \leftarrow r(T,U)$



# Eliminating Existential Variables via U/F

D1:  $r(L,U) \leftarrow X > U \wedge \text{list}(L) \wedge \text{member}(X,L)$

Unfold:  $r([X|T],U) \leftarrow X > U \wedge \text{list}(T)$

$r([X|T],U) \leftarrow Y > U \wedge \text{list}(T) \wedge \text{member}(Y,T)$

Fold:  $r([X|T],U) \leftarrow X > U \wedge \text{list}(T)$

$r([X|T],U) \leftarrow r(T,U)$

} LR-clauses  
(no existential variables)



# Transformed program (1)

D4:  $\text{prop} \leftarrow \neg p$

D3:  $p \leftarrow \text{list}(L) \wedge \neg q(L)$

D2:  $q(L) \leftarrow \text{list}(L) \wedge \neg r(L, U)$

D1:  $r([X|T], U) \leftarrow X > U \wedge \text{list}(T)$   
 $r([X|T], U) \leftarrow r(T, U)$

No existential variables

Note that the new definition of the predicate  $r$  is used for unfolding

# *Introducing New Definitions*

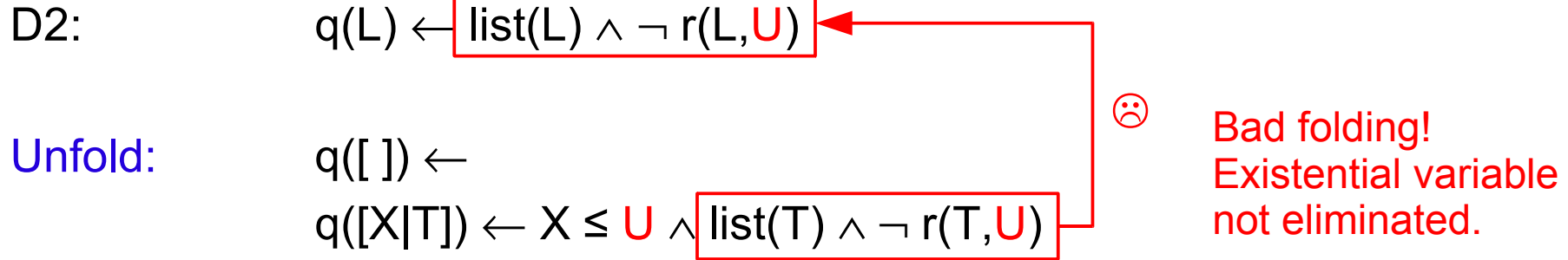
D2:  $q(L) \leftarrow \text{list}(L) \wedge \neg r(L, U)$

# Introducing New Definitions

D2:  $q(L) \leftarrow \underline{\text{list}(L)} \wedge \underline{\neg r(L, U)}$

Unfold:  $q([ ]) \leftarrow$   
 $q([X|T]) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T, U)$

# Introducing New Definitions



# Introducing New Definitions

D2:  $q(L) \leftarrow \text{list}(L) \wedge \neg r(L, U)$

Unfold:  $q([ ]) \leftarrow$

$q([X|T]) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T, U)$

Define:

$q1(X, T) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T, U)$

# Introducing New Definitions

D2:  $q(L) \leftarrow \text{list}(L) \wedge \neg r(L, U)$

Unfold:  $q([ ]) \leftarrow$

$q([X|T]) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T, U)$

Define:

$q1(X, T) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T, U)$

Fold:

$q([ ]) \leftarrow$

$q([X|T]) \leftarrow q1(X, T)$

} LR-clauses 😊  
(no existential variables)

Existential variables to be eliminated from the new definition

# *Transforming New Definitions*

We transform the new definition into a set of LR-clauses

New Def.:  $q1(X,T) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

# Transforming New Definitions

We transform the new definition into a set of LR-clauses

New Def.:  $q1(X,T) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

Unfold:  $q1(X,[ ]) \leftarrow$

$q1(X,[Y|T]) \leftarrow X \leq U \wedge Y \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$



Bad folding!  
Existential variable  
not eliminated

# Transforming New Definitions

We transform the new definition into a set of LR-clauses

New Def.:  $q1(X,T) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

Unfold:  $q1(X,[ ]) \leftarrow$

$q1(X,[Y|T]) \leftarrow X \leq U \wedge Y \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$  linear order

$\equiv (X > Y \wedge X \leq U) \wedge (X \leq Y \wedge Y \leq U)$

Replace-constraints:

$q1(X,[Y|T]) \leftarrow X > Y \wedge X \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

$q1(X,[Y|T]) \leftarrow X \leq Y \wedge Y \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

# Transforming New Definitions

We transform the new definition into a set of LR-clauses

New Def.:  $q1(X,T) \leftarrow X \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

Unfold:  $q1(X,[ ]) \leftarrow$

$q1(X,[Y|T]) \leftarrow X \leq U \wedge Y \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

Replace-constraints:

$q1(X,[Y|T]) \leftarrow X > Y \wedge X \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

$q1(X,[Y|T]) \leftarrow X \leq Y \wedge Y \leq U \wedge \text{list}(T) \wedge \neg r(T,U)$

Fold:  $q1(X,[ ]) \leftarrow$

$q1(X,[Y|T]) \leftarrow X > Y \wedge q1(X,T)$

$q1(X,[Y|T]) \leftarrow X \leq Y \wedge q1(Y,T)$

LR-clauses  
(no existential variables)

# Transformed program (2)

D4:  $f \leftarrow \neg p$

D3:  $p \leftarrow \text{list}(L) \wedge \neg q(L)$

D2:  $q([\ ])$   $\leftarrow$   
 $q([X|T]) \leftarrow q1(X,T)$   
 $q1(X,[Y|T]) \leftarrow X > Y \wedge q1(X, L)$   
 $q1(X,[Y|T]) \leftarrow X \leq Y \wedge q1(Y, L)$

No existential variables

D1:  $r([X|T],U) \leftarrow X > U \wedge \text{list}(T)$   
 $r([X|T],U) \leftarrow r(T,U)$

# *Deriving a Propositional Program*

D3:  $p \leftarrow \text{list}(L) \wedge \neg q(L)$

# Deriving a Propositional Program

D3:  $p \leftarrow \underline{\text{list(L)}} \wedge \neg \underline{q(L)}$

Unfold:  $p \leftarrow \text{list(T)} \wedge \neg q1(X,T)$       Folding not possible

# Deriving a Propositional Program

D3:  $p \leftarrow \text{list}(L) \wedge \neg q(L)$

Unfold:  $p \leftarrow \boxed{\text{list}(T) \wedge \neg q1(X, T)}$



Define:  $p1 \leftarrow \boxed{\text{list}(T) \wedge \neg q1(X, T)}$

# Deriving a Propositional Program

D3:  $p \leftarrow \text{list}(L) \wedge \neg q(L)$

Unfold:  $p \leftarrow \boxed{\text{list}(T) \wedge \neg q1(X, T)}$

Define:  $p1 \leftarrow \boxed{\text{list}(T) \wedge \neg q1(X, T)}$

Fold:  $p \leftarrow \boxed{p1}$       LR-clause  
(no existential variables,  
propositional)

# *Deriving a Propositional Program*

New Def.:  $p1 \leftarrow \text{list}(T) \wedge \neg q1(X, T)$

# Deriving a Propositional Program

New Def.:  $p1 \leftarrow \underline{\text{list}(T)} \wedge \underline{\neg q1(X,T)}$

Unfold:  $p1 \leftarrow X > Y \wedge \text{list}(T) \wedge \neg q1(X,T)$

$p1 \leftarrow X \leq Y \wedge \text{list}(T) \wedge \neg q1(Y,T)$

# Deriving a Propositional Program

New Def.:  $p1 \leftarrow \text{list}(T) \wedge \neg q1(X, T)$

Unfold:  $p1 \leftarrow X > Y \wedge \text{list}(T) \wedge \neg q1(X, T)$

$p1 \leftarrow X \leq Y \wedge \text{list}(T) \wedge \neg q1(Y, T)$

$\exists Y X > Y \equiv \text{true}$

$\exists X X \leq Y \equiv \text{true}$

Replace-Constraints (variable elimination):

$p1 \leftarrow \text{list}(T) \wedge \neg q1(Y, T)$

# Deriving a Propositional Program

New Def.:

$$p1 \leftarrow \boxed{\text{list}(T) \wedge \neg q1(X, T)}$$

Unfold:

$$p1 \leftarrow \boxed{X > Y} \wedge \text{list}(T) \wedge \neg q1(X, T)$$

$$p1 \leftarrow \boxed{X \leq Y} \wedge \text{list}(T) \wedge \neg q1(Y, T)$$

Replace-Constraints (variable elimination):

$$p1 \leftarrow \boxed{\text{list}(T) \wedge \neg q1(Y, T)}$$

Fold:

$$p1 \leftarrow \boxed{p1}$$

LR-clause  
(no existential variables,  
propositional)



# The Final LR-Program

Prop: No existential variables

T {

- D4:  $\text{prop} \leftarrow \neg p$
- D3:  $p \leftarrow p1$   
 $p1 \leftarrow p1$
- D2:  $q([\ ])$   $\leftarrow$   
 $q([X|T]) \leftarrow q1(X, T)$   
 $q1(X, [Y|T]) \leftarrow X > Y \wedge q1(X, L)$   
 $q1(X, [Y|T]) \leftarrow X \leq Y \wedge q1(Y, L)$
- D1:  $r([X|T], U) \leftarrow X > U \wedge \text{list}(T)$   
 $r([X|T], U) \leftarrow r(T, U)$

$M(\text{Prop}) = \{\text{prop}\} \Rightarrow M(P) \models \varphi$

# Experiments

Some example runs using the MAP transformation system  
([www.iasi.cnr.it/~proietti/system.html](http://www.iasi.cnr.it/~proietti/system.html))

Constraints are handled using the `clp(R)` module of SICStus Prolog  
(implementing a variant of Fourier-Motzkin variable elimination)

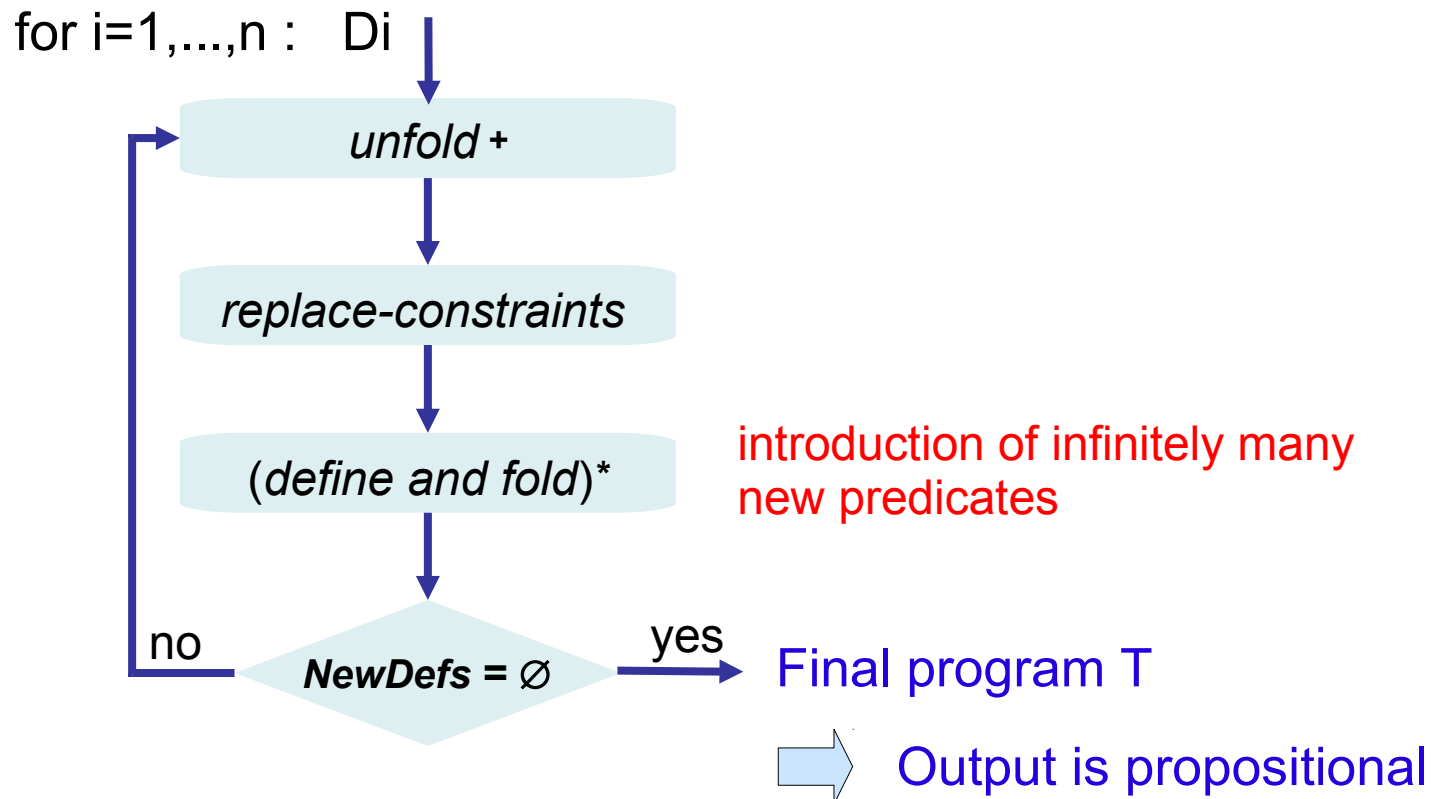
Proven formulas in the theory of linear orders, lists, and addition

Property	Time (PM 1.73)
$\forall L \exists U \forall Y ( \text{member}(Y,L) \rightarrow Y \leq U )$	31 ms
$\forall L \forall Y ( (\text{sumlist}(L,Y) \wedge Y > 0) \rightarrow \exists X (\text{member}(X,L) \rightarrow X > 0) )$	15 ms
$\forall L \forall M \forall N ( (\text{ord}(L) \wedge \text{ord}(M) \wedge \text{sumzip}(L,M,N)) \rightarrow \text{ord}(N) )$	16 ms
$\forall L \forall M \forall X \forall Y ( (\text{leqlist}(L,M) \wedge \text{sumlist}(L,X) \wedge \text{sumlist}(M,Y)) \rightarrow X \leq Y )$	16 ms

[PPS06] (ICLP)

# Termination of the Unfold-Fold Strategy

The only source for **nontermination** is the possible introduction of infinitely many new predicates.

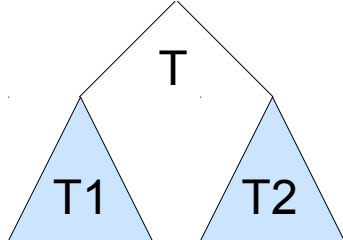


# Again on Generating Trees and Testing

tree(e).

tree(t(L,R))  $\leftarrow$  tree(L)  $\wedge$  tree(R).

balanced(T)  $\equiv$  tree(T)  $\wedge$   $\forall$  D1,D2 ( depth(T,D1)  $\wedge$  depth(T,D1)  $\rightarrow$  D1=D2 )

merge(T1,T2,T)  $\equiv$   only if T is balanced

Generate test trees T1 and T2 and check whether T is balanced.

Consider only 'interesting' trees (e.g. don't generate isomorphic trees).

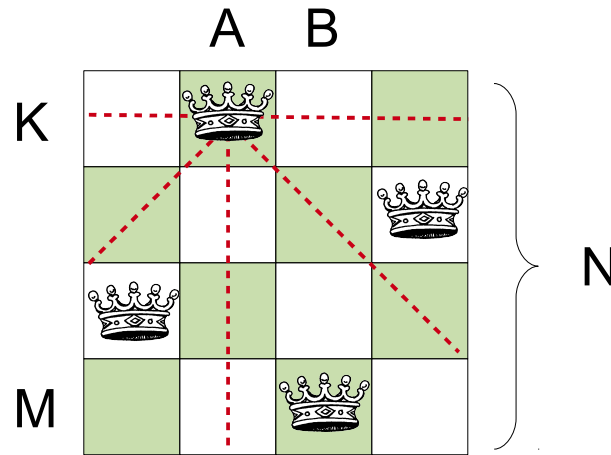
Restricting the class of LP used for describing 'types' and operations on them one can develop a complete proof method for properties like:

prop  $\leftarrow$   $\forall$  T,T1,T2 ( tree(T1)  $\wedge$  tree(T2)  $\wedge$  merge(T1,T2,T)  $\rightarrow$  balanced(T) )

# *Synthesis by Program Transformation*

# The Transformational Synthesis Method

N-queens :



A First Order specification :

$$\begin{aligned} \text{board}(\mathbf{N}, \mathbf{L}) &\stackrel{\text{def}}{=} \text{nat}(\mathbf{N}) \wedge \text{nat\_list}(\mathbf{L}) \wedge \text{length}(\mathbf{L}, \mathbf{N}) \wedge \\ &\quad \forall X (\text{member}(X, \mathbf{L}) \rightarrow 1 \leq X \leq \mathbf{N}) \wedge \\ &\quad \forall A \forall B \forall K \forall M (1 \leq K < M \wedge \text{occurs}(A, K, \mathbf{L}) \wedge \text{occurs}(B, M, \mathbf{L}) \\ &\quad \rightarrow (A \neq B \wedge A - B \neq M - K \wedge B - A \neq M - K)) \end{aligned}$$

# The Transformational Synthesis Method

## Step 1. LT-transformation

$$\text{board}(N,L) \leftarrow \underbrace{\text{nat}(N) \wedge \text{nat\_list}(L) \wedge \text{length}(L,N)}_{\text{generate a list } L} \wedge \underbrace{\neg \text{new1}(L,N) \wedge \neg \text{new2}(L)}_{\text{test if } L \text{ is good}}$$
$$\text{new1}(N,L) \leftarrow \text{member}(X,L) \wedge \neg (1 \leq X \leq N)$$
$$\text{new2}(L) \leftarrow 1 \leq K < M \wedge \neg (A \neq B \wedge A-B \neq M-K \wedge B-A \neq M-K) \wedge \text{occurs}(A,K,L) \wedge \text{occurs}(B,M,L)$$

...a very bad program

# The Transformational Synthesis Method

## Step 1. LT-transformation

$$\text{board}(N,L) \leftarrow \underbrace{\text{nat}(N) \wedge \text{nat\_list}(L) \wedge \text{length}(L,N)}_{\text{generate a list } L} \wedge \underbrace{\neg \text{new1}(L,N) \wedge \neg \text{new2}(L)}_{\text{test if } L \text{ is good}}$$
$$\text{new1}(N,L) \leftarrow \text{member}(X,L) \wedge \neg (1 \leq X \leq N)$$
$$\text{new2}(L) \leftarrow 1 \leq K < M \wedge \neg (A \neq B \wedge A-B \neq M-K \wedge B-A \neq M-K) \wedge \text{occurs}(A,K,L) \wedge \text{occurs}(B,M,L)$$

...a very bad program

## Step 2. Elimination of Existential Variables

$$\text{board}(N,L) \leftarrow \text{new3}(N,L,0)$$
$$\text{new3}(N,[],K) \leftarrow N = K$$
$$\text{new3}(N,[H|T],K) \leftarrow N \geq K+1 \wedge \text{new3}(N,T,K+1) \wedge \text{new4}(H,T,N,0)$$
$$\text{new4}(A,[],N,M) \leftarrow 1 \leq A \leq N$$
$$\text{new4}(A,[B|T],N,M) \leftarrow A \neq B \wedge A-B \neq M+1 \wedge B-A \neq M+1 \wedge \text{new4}(A,T,N,M+1)$$

The new program:

- is **incremental**, i.e. places one queen at a time
- uses **backtracking**
- has **no exist. variables**
- **terminates** for any given  $n$

End of the story...

Thank you !

