

Program Specialization for Verifying Infinite State Systems: An Experimental Evaluation

Valerio Senni

LORIA-INRIA

&

U. of Rome Tor Vergata, Italy

joint work with

Fabio Fioravanti, U. of Chieti-Pescara, Italy

Alberto Pettorossi, U. of Rome Tor Vergata, Italy

Maurizio Proietti IASI-CNR, Rome, Italy



ERCIM

European Research Consortium
for Informatics and Mathematics

Outline

Verification of infinite state systems

- Computation Tree Logic
- Constraint Logic Programming

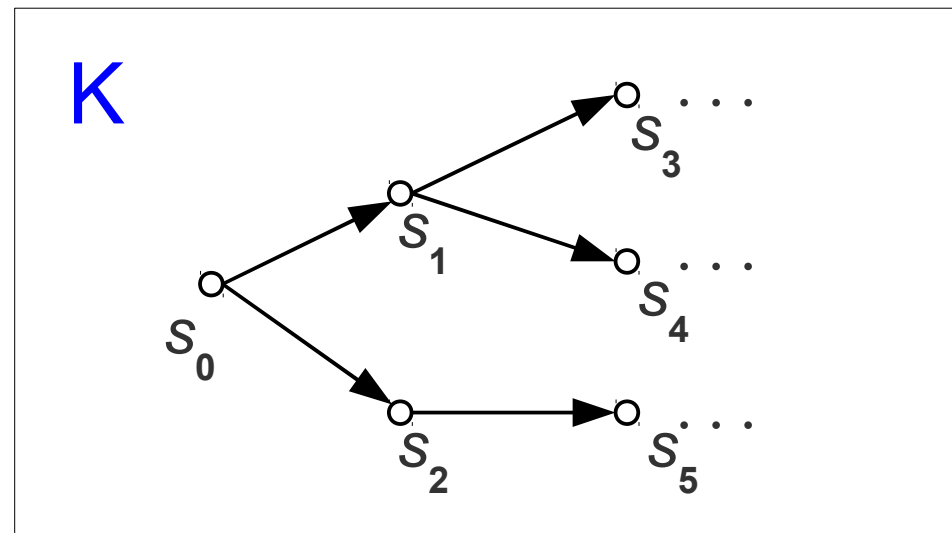
Two-phase Verification method

- Rule-based Program Specialization
 - Termination using generalization techniques
- Perfect Model Computation

Experimental evaluation

Infinite-state transition systems

- Concurrency
- Nondeterminism
- States
- Transitions



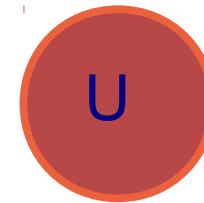
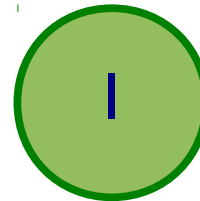
Executions are infinite branches
of the computation tree

Example: checking reachability

$K = \langle S, T \rangle$

I initial states

U unsafe states



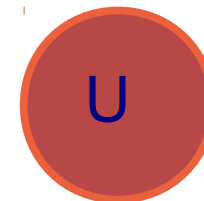
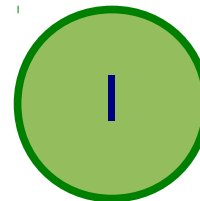
Forward

Problems:

- Convergence

- Precision

(competing issues)



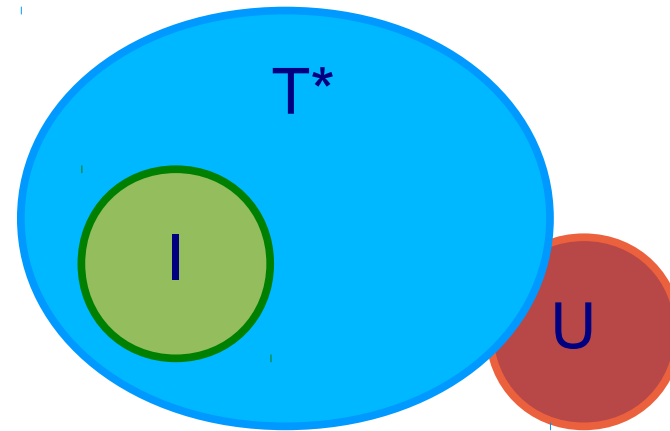
Backward

Example: checking reachability

$K = \langle S, T \rangle$

I initial states

U unsafe states

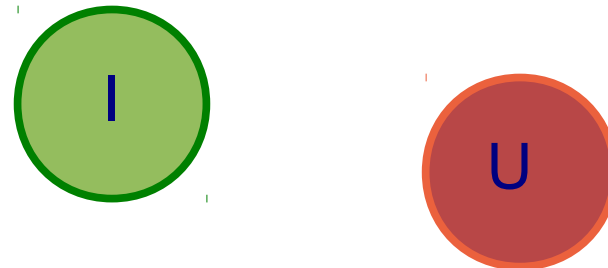


Forward

Problems:

- Convergence
- Precision

(competing issues)



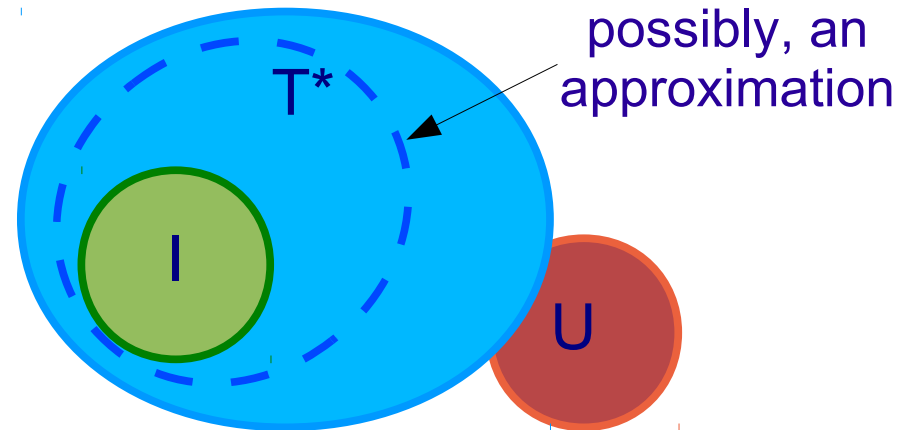
Backward

Example: checking reachability

$K = \langle S, T \rangle$

I initial states

U unsafe states

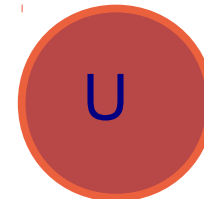
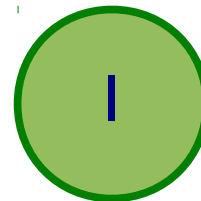


Forward

Problems:

- Convergence
- Precision

(competing issues)



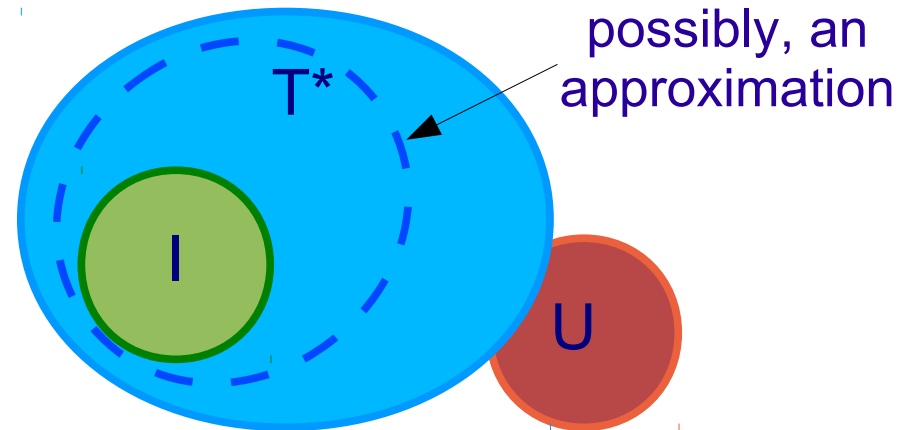
Backward

Example: checking reachability

$K = \langle S, T \rangle$

I initial states

U unsafe states

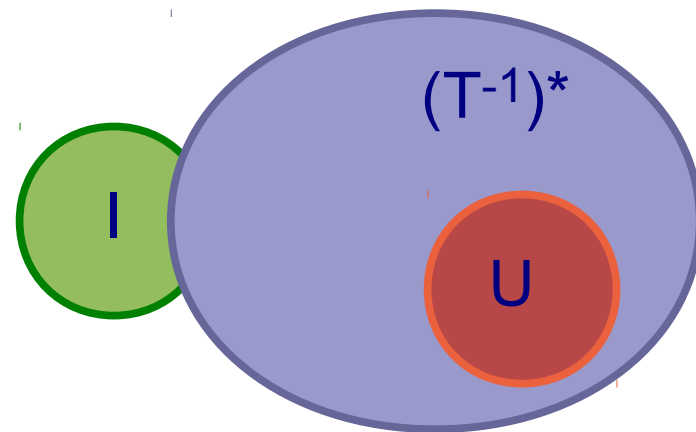


Forward

Problems:

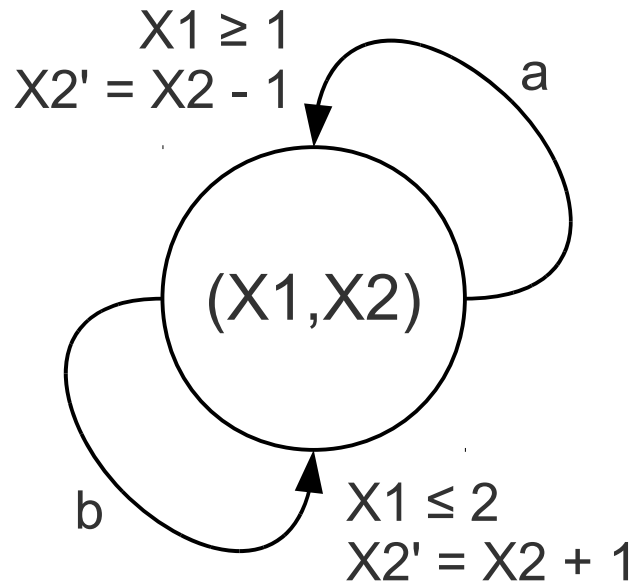
- Convergence
- Precision

(competing issues)



Backward

A concrete example



S : $(X1, X2)$ over \mathbb{Z}

T : $\{a, b\}$

I : $X1 \leq 0 \wedge X2 = 0$

U : $X2 < 0$

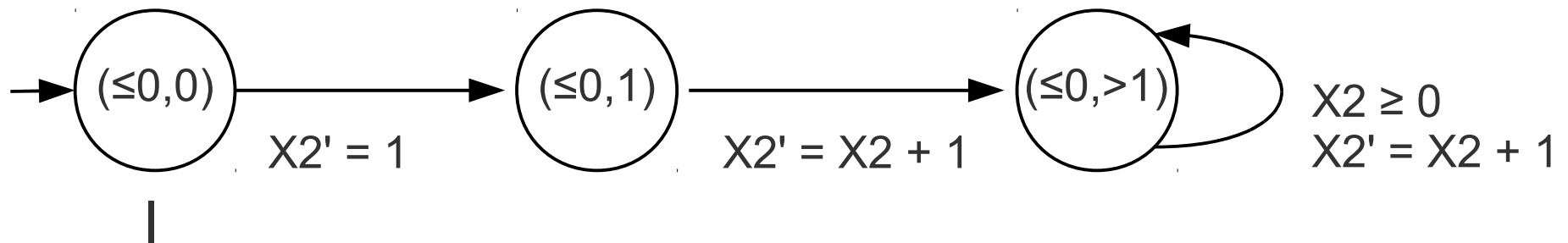
To show: $T^*(I) \cap U = \emptyset$

Very simple system, but several tools fail to prove the property:

- ALV [Bultan] loops or returns 'unable to prove'
- DMC [Delzanno] loops (backward only)
- HyTech [Henzinger] loops

Our two-step method

(1) Specialization wrt the initial states (forward)



A new system that encodes only the states *reachable* from $|$
We have *no loss of precision* in this phase.

(2) Backward analysis

After (1) it is much easier to check that in any state $X2 < 0$

Computational Tree Logic

Properties expressed in CTL,
a propositional logic augmented with:

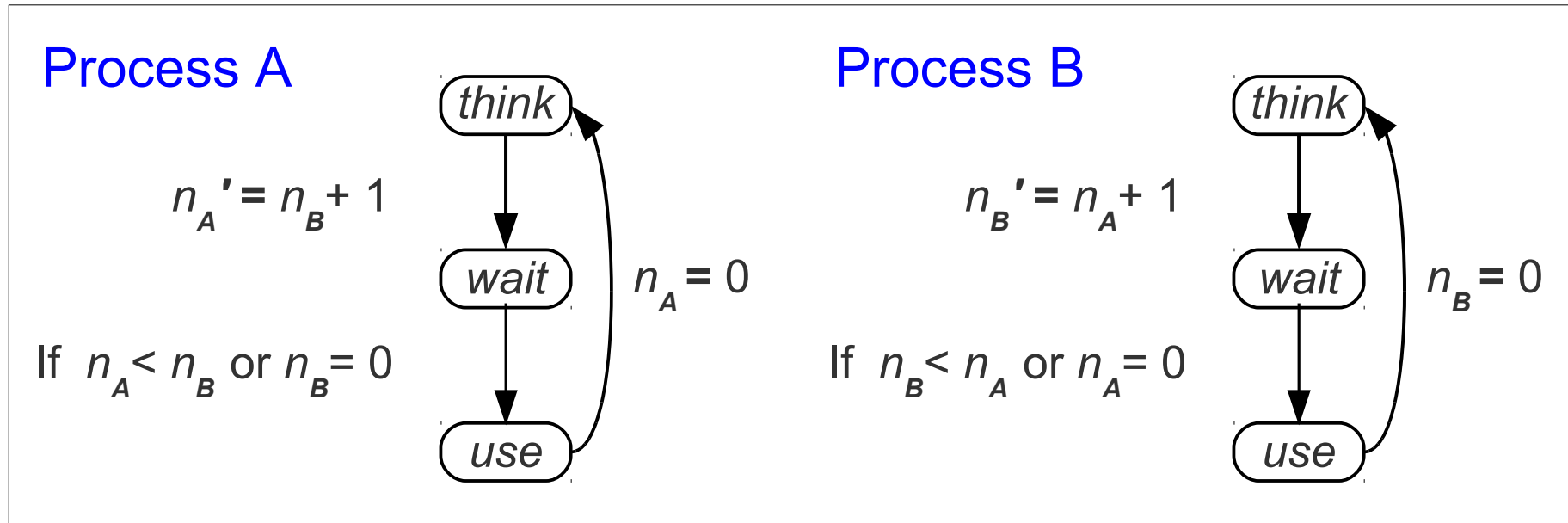
- quantifiers over nondeterminism: **E** (Exists), **A** (All)
- temporal operators over executions:
 - X** (Next, in the next state in the path),
 - F** (Future, there exists a state in the path),
 - G** (Globally, for all states of the path).

CTL Model Checking: decide whether or not $K, s \models \varphi$

- decidable for finite state systems (polynomial time)
- undecidable for infinite state systems

The Bakery Protocol (Lamport)

Each process has: control state: $s \in \{think, wait, use\}$ and counter: $n \in N$



System: $A \parallel B$

Path: $\langle think, 0, think, 0 \rangle \rightarrow \langle wait, 1, think, 0 \rangle \rightarrow \langle wait, 1, wait, 2 \rangle \rightarrow \langle use, 1, wait, 2 \rangle \rightarrow \dots$

Mutual Exclusion: $\langle think, 0, think, 0 \rangle \models \neg EF \text{ unsafe}$

where, for all n_A, n_B : $\langle use, n_A, use, n_B \rangle \models \text{unsafe}$

Encoding as a Constraint Logic Program (1)

Encoding of the transition system $\langle S, T \rangle$

- the transition relation is encoded by a binary predicate

$$\text{tr}(\langle \text{think}, A, S, B \rangle, \langle \text{wait}, A1, S, B \rangle) \leftarrow A1 = B + 1$$

$$\text{tr}(\langle \text{wait}, A, S, B \rangle, \langle \text{use}, A, S, B \rangle) \leftarrow A < B$$

$$\text{tr}(\langle \text{wait}, A, S, B \rangle, \langle \text{use}, A, S, B \rangle) \leftarrow B = 0$$

$$\text{tr}(\langle \text{use}, A, S, B \rangle, \langle \text{think}, A1, S, B \rangle) \leftarrow A1 = 0$$

+ similar clauses for process B

- initial states: $\text{initial}(\langle \text{think}, A, \text{think}, B \rangle) \leftarrow A = 0, B = 0$

- elementary properties: $\text{elem}(\langle \text{use}, A, \text{use}, B \rangle, \text{unsafe}) \leftarrow$

P_K

Encoding as a Constraint Logic Program (2)

Encoding of the satisfaction relation $K, s \models \varphi$

P_{sat} {

- sat(X, F) \leftarrow elem(X,F)
- sat(X, not(F)) \leftarrow \neg sat(X,F)
- sat(X, and(F1,F2)) \leftarrow sat(X,F1) \wedge sat(X,F2)
- sat(X, ex(F)) \leftarrow tr(X,Y) \wedge sat(Y,F)
- sat(X, eu(F1,F2)) \leftarrow sat(X,F2)
- sat(X, eu(F1,F2)) \leftarrow sat(X,F1) \wedge tr(X,Y) \wedge sat(Y,eu(F1,F2))
- sat(X, af(F)) \leftarrow sat(X,F)
- sat(X, af(F)) \leftarrow ts(X,Ys) \wedge sat_all(Ys,af(F))
- sat_all([],F) \leftarrow
- sat_all([X|Xs],F) \leftarrow sat(X,F) \wedge sat_all(Xs,F)

Encoding as a Constraint Logic Program (3)

Encoding of the property to be verified

$$\text{prop} \equiv \text{def } \forall X (\text{initial}(X) \rightarrow \text{sat}(X, \varphi))$$

by Lloyd-Topor transformation:

$$P_{\text{prop}} \left\{ \begin{array}{l} \text{prop} \leftarrow \neg \text{negprop} \\ \text{negprop} \leftarrow \text{initial}(X), \neg \text{sat}(X, \varphi) \end{array} \right.$$

The program encoding our Model Checking problem is:

$$P = P_K \cup P_{\text{sat}} \cup P_{\text{prop}}$$

Correctness of the Encoding

$$P = P_K \cup P_{\text{sat}} \cup P_{\text{prop}}$$

P is locally stratified, and thus it has a *unique perfect model*

Theorem 1. Let K be a Kripke structure, let I be the set of initial states of K , and let φ be a CTL formula. Then,
(for all states $s \in I$, $K, s \models \varphi$) iff $\text{prop} \in M(P)$

But...

Bottom-up construction of $M(P)$ from facts may not terminate because $M(P)$ is infinite.

Top-down evaluation of P from prop may not terminate due to infinite computation paths.

Two-phase Verification Method

Phase 1: specialize P w.r.t. the query prop :

$$P \Rightarrow \dots \Rightarrow SP \quad \text{s.t.} \quad \text{prop} \in M(P) \text{ iff } \text{prop} \in M(SP)$$

and keep only the clauses on which the predicate prop depends. SP is a *stratified* program.

Specialization is performed by using the rules + strategies program transformation approach.

(' \Rightarrow ') is an application of a transformation rule)

Phase 2: construct bottom-up the perfect model of $M(SP)$
(may not terminate)

Transformation Rules

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

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

Unfolding

\Rightarrow

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

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

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

Folding

\Rightarrow

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

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

Goal

Replacement

\Rightarrow

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

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

Few, general rules

Dependent on the context

Specialization strategy

Input: The program P

Output : A stratified program SP s.t.

$\text{prop} \in M(P)$ iff $\text{prop} \in M(SP)$

C1. $\text{prop} \leftarrow \neg \text{negprop}$

C2. $\text{negprop} \leftarrow \text{initial}(X), \neg \text{sat}(X, \phi)$

recall

$SP := \{C1\}; \text{ InDefs} := \{C2\}; \text{ Defs} := \{ \};$

while (there exists a clause γ in InDefs) **do**

Unfold(γ, Γ);

Generalize&Fold($\text{Defs}, \Gamma, \text{NewDefs}, \Phi$);

$SP := SP \cup \Phi; \text{ InDefs} := (\text{InDefs} - \{\gamma\}) \cup \text{NewDefs};$

end-while

Termination of specialization (Phase 1)

Local control

- Termination of the Unfold procedure

Global control

- Termination of the while loop
- We use constraint generalization techniques

Generalization

1. For limiting the number of clauses introduced by definition, sometimes we introduce definitions containing a generalized constraint
2. **When? Firing Relations (well quasi orderings).** Ensure generalization is eventually applied
3. **How? Generalization Operators.** Each definition can be generalized a finite number of times only
4. Selecting a good **generalization strategy** is not trivial
 - Too coarse → unable to prove property
 - Too fine-grained → high verification times

Introduction of new Definitions

Goal: fold any (non-unit, unfolded) clause $\gamma \in \Gamma$

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

$$\delta: \mathbf{K} \leftarrow a \wedge \mathbf{B} \quad d \sqsubseteq a \text{ (entailment)}$$

Three cases, for a given γ :

1. can be folded (no new definitions)
2. cannot be folded, no *ancestor* of γ is in the firing relation

$$\delta: \mathbf{K} \leftarrow d \wedge \mathbf{B} \quad d \sqsubseteq d \quad \text{(no generalization)}$$

3. cannot be folded, an *ancestor* of γ is in the firing relation

$$\delta: \mathbf{K} \leftarrow c\Theta d \wedge \mathbf{B} \quad d \sqsubseteq c\Theta d \quad \text{(generalization)}$$

Termination: any sequence $(c_0 \Theta d_0), (c_1 \Theta d_1), \dots$ eventually **stabilizes**

The constraint domain Lin_k

1. Lin_k are linear inequations over k distinct variables X_1, \dots, X_k
2. Constraints of Lin_k are conjunctions of atomic constraints of the form

$$p \leq 0 \text{ or } p < 0$$

where p is a polynomial of the form

$$q_0 + q_1 X_1 + \dots + q_k X_k$$

and q_i 's are integers

Well-quasi orderings

A well-quasi ordering on a set S is a reflexive, transitive, binary relation \leq such that,

for every infinite sequence e_0, e_1, \dots of elements of S , there exist i and j s.t. $i < j$ and $e_i \leq e_j$

e.g.

MaxCoeff compares the maximum absolute value of coefficients:

for any two atomic constraints q and r , we have that $q \leq r$ iff $\max\{|q_0|, \dots, |q_k|\} \leq \max\{|r_0|, \dots, |r_k|\}$

Generalization operators (1)

Given a $wqo \preceq$, the generalization of a constraint c w.r.t. a constraint d is a constraint $c \ominus d$ such that

$$- d \sqsubseteq c \ominus d \quad (i)$$

$$- c \ominus d \preceq c \quad (ii)$$

(i) $c \ominus d$ can replace d in a candidate definition for folding

(ii) every infinite sequence of constraints constructed by using the generalization operator eventually stabilizes

(similar to the widening operator in abstract interpretation)

In general, \ominus is not commutative

Generalization operators (2)

Let $c = c_1, \dots, c_m$ and $d = d_1, \dots, d_n$

1. **Top**: $c \Theta d$ is the constraint *true*
2. **Widen**: $c \Theta d$ is the conjunction of all c_i 's such that $d \sqsubseteq c_i$
3. **WidenPlus**: $c \Theta d$ is the conjunction of all c_i 's such that $d \sqsubseteq c_i$ and of all d_j 's such that $d_j \leq c$
4. **CHWiden** and **CHWidenPlus** are obtained by applying the Convex Hull operator

Experimental evaluation

Experiments performed using the MAP transformation system

<http://www.iasi.cnr.it/~proietti/system.html>

Mutual exclusion protocols

Parameterized cache coherence protocols

Other systems (train controllers, client/server, sorting algorithms, petri nets)

Comparison of generalization operators

	T	W	CHM	CHS	$CHWM$	$CHWS$	WM	WS
Bakery2 (safety)	80	150	30	30	40	50	30	20
Bakery2 (liveness)	∞	∞	110	100	130	120	90	60
Bakery3 (safety)	3940	4900	430	430	460	460	170	170
MutAst	2330	320	390	400	420	440	80	160
Peterson	∞	∞	860	870	1380	1410	190	220
Ticket (safety)	20	30	30	20	20	10	20	20
Ticket (liveness)	110	100	100	100	80	90	80	110
Berkeley RISC	30	30	180	170	210	200	30	30
DEC Firefly	70	130	200	130	310	320	20	30
IEEE Futurebus+	16380	47570	∞	47120	75860	47630	110	2460
Illinois University	100	70	50	60	50	50	10	30
MESI	70	40	250	250	130	130	30	30
MOESI	100	150	330	170	120	170	40	60
Synapse N+1	10	20	10	30	30	30	20	20
Xerox PARC Dragon	50	60	220	220	280	270	30	30
Barber	∞	28440	2000	2050	2530	2560	1160	1220
Bounded Buffer	30	360	9490	9570	5800	5840	3580	3580
Unbounded Buffer	∞	∞	410	400	420	420	3810	3810
Consprodjava	∞	∞	∞	∞	∞	∞	25300	∞
CSM	∞	∞	3820	3880	4830	4860	6410	6540
Consistency v1	∞	∞	410	450	780	780	70	60
Consistency v2	∞	70	110	130	220	260	40	60
Insertion Sort	80	70	130	120	160	170	100	90
Selection Sort	∞	∞	∞	160	230	180	∞	180
Office Light Control	50	40	50	50	50	50	50	50
Reset Petri Net	∞	∞	∞	∞	∞	∞	20	20
Kanban	∞	∞	15630	15800	17790	18040	8130	8000
Train	∞	1440	3420	6290	3680	6650	30900	57260

28 systems

Experiments run on a Linux Intel Core 2, 2.66GHz machine

Table 3. *Verification times for the MAP system.* For each example we show the *total verification time* (Phases 1 and 2) obtained by using the firing relation *Always* in conjunction with the generalization operators: \ominus_T , \ominus_W , \ominus_{CHM} , \ominus_{CHS} , \ominus_{CHWM} , \ominus_{CHWS} , \ominus_{WM} , and \ominus_{WS} . Times are expressed in milliseconds (ms).

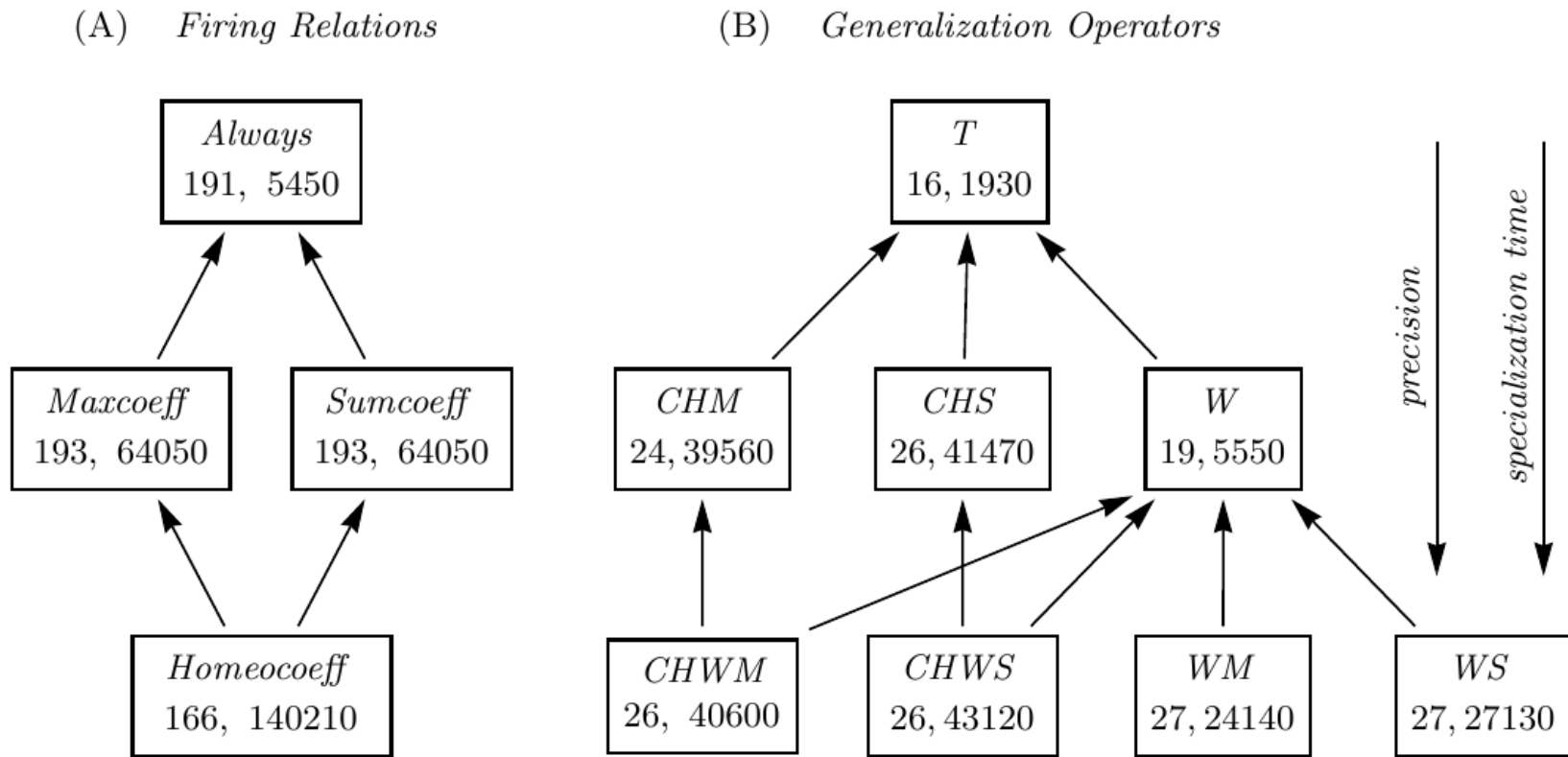


Figure 3. Comparison of firing relations and generalization operators. (A) For each firing relation the numbers on the left and on the right indicate, respectively: (i) the number of properties verified by using that firing relation in conjunction with all generalization operators, and (ii) the sum of the specialization times taken by using that firing relation in conjunction with all generalization operators. (B) For each generalization operator the numbers on the left and on the right indicate, respectively: (i) the total number of properties verified (see Table 3), and (ii) the sum of the specialization times (see Table 4).

Note: HomeoCoeff suffers from timeout/out-of-memory

Comparison with other systems

Action Language Verifier [Bultan]

- combines BDD-based symbolic manipulation for boolean and enumerated types, with a solver for linear constraints on integers

DMC [Delzanno]

- computes (approximated) least and greatest models of CLP(R) programs

HyTech [Henzinger]

- model checker for hybrid systems

Comparison with other MC Tools

EXAMPLE	MAP		ALV				DMC		HyTech	
	WM	WS	default	A	F	L	noAbs	Abs	Fw	Bw
Bakery 2 (safety)	30	20	20	30	90	30	10	30	∞	20
Bakery 2 (liveness)	90	60	30	30	90	30	60	70	\times	\times
Bakery 3 (safety)	170	170	580	570	∞	600	460	3090	∞	360
MutAst	80	160	\perp	\perp	910	\perp	150	1370	70	130
Peterson N	190	220	71690	\perp	∞	∞	∞	∞	70	∞
Ticket (safety)	20	20	∞	80	30	∞	∞	60	∞	∞
Ticket (liveness)	80	110	∞	230	40	∞	∞	220	\times	\times
Berkeley RISC	30	30	10	\perp	20	60	30	30	∞	20
DEC Firefly	20	20	10	\perp	20	80	50	80	∞	20
IEEE Futurebus+	110	2460	320	\perp	∞	670	4670	9890	∞	380
Illinois University	10	30	10	\perp	∞	140	70	110	∞	20
MESI	30	30	10	\perp	20	60	40	60	∞	20
MOESI	40	60	10	\perp	40	100	50	90	∞	10
Synapse N+1	20	20	10	\perp	10	30	0	0	∞	0
Xerox PARC Dragon	30	30	20	\perp	40	340	70	120	∞	20
Barber	1160	1220	340	\perp	90	360	140	230	∞	90
Bounded Buffer	3580	3580	0	10	∞	20	20	30	∞	10
Unbounded Buffer	3810	3810	10	10	40	40	∞	∞	∞	20
Consprodjava	25300	∞	∞	∞	∞	∞	∞	∞	∞	∞
CSM	6410	6540	79490	\perp	∞	∞	∞	∞	∞	∞
Consistency v1	70	60	∞	\perp	∞	∞	∞	∞	∞	2030
Consistency v2	40	60	∞	\perp	40	∞	∞	∞	∞	2790
Insertion Sort	100	90	40	60	∞	70	30	80	∞	10
Selection Sort	∞	180	∞	390	∞	∞	∞	∞	∞	∞
Office Light Control	50	50	20	20	30	20	10	10	∞	∞
Reset Petri Nets	20	20	∞	\perp	∞	10	0	0	∞	10
Kanban	8130	8000	∞	∞	∞	∞	∞	∞	700	∞
Train	30900	57260	10	\perp	∞	30	∞	∞	∞	∞

Experiments run on a Linux Intel Core 2, 2.66GHz machine

Analysis

Precision (number of properties proved) and average verification time

MAP	27/28	(894 ms)	
DMC (with abstraction)	19/28	(928 ms)	[95 ms]
ALV (default option)	19/28	(8033 ms)	[2462 ms]
HyTech (backwards)	18/28	(331 ms)	[519 ms]

Analysis

Bounded and Unbounded Buffer can be easily verified by backward reachability

- The specialization phase is redundant
- MAP slower than other systems

Peterson, Consprodjava, and CSM examples

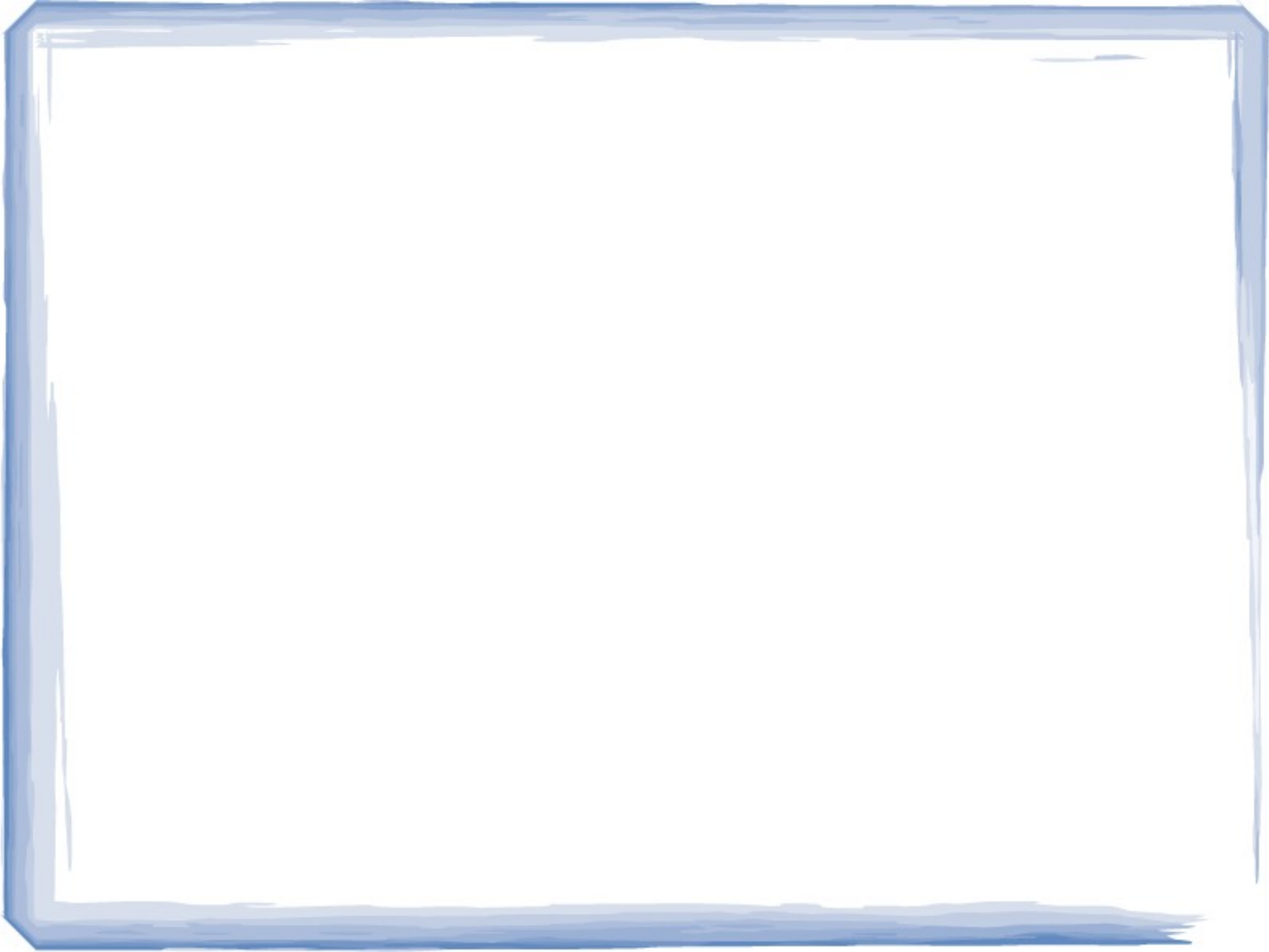
- The specialization phase pays off
- MAP much more efficient than other systems

Future work

Use approximation methods during the bottom-up computation of the perfect model (Phase 2)

Apply specialization to concurrent systems specified in different languages, not necessarily (C)LP based

The end



HomeoCoeff wqo

HomeoCoeff compares sequences of absolute values of integer coefficients

$$q_0 + q_1 X_1 + \dots + q_k X_k \leq r_0 + r_1 X_1 + \dots + r_k X_k \quad (i)$$

iff there exist a permutation h of the indexes $\langle 0, \dots, k \rangle$ such that, for $i=0, \dots, k$ $|q_i| \leq |r_{h(i)}|$

Extended to atomic constraints and constraints, for example $q < 0 \leq r < 0$ iff (i) holds

MaxCoeff and SumCoeff wqo's

MaxCoeff compares the maximum absolute value of coefficients

for any two atomic constraints q and r , we have that $q \leq r$ iff $\max\{|q_0|, \dots, |q_k|\} \leq \max\{|r_0|, \dots, |r_k|\}$

SumCoeff compares the sum of the absolute value of coefficients

Similarly $q \leq r$ iff $|q_0| + \dots + |q_k| \leq |r_0| + \dots + |r_k|$

Experimental evaluation

Parameterized cache coherence protocols

- Berkeley RISC, DEC Firefly, IEEE Futurebus+, Illinois University, MESI, MOESI, Synapse N+1, and Xerox PARC Dragon.

Used in shared-memory multiprocessing systems for guaranteeing data consistency of the local cache associated with every CPU

Experimental evaluation

Other systems

- Parameterized barber problem with N customers
- Producer-consumer via Bounded and Unbounded buffer
- CSM: a central server model
- Insertion and selection sort: check array bounds
- Office light control
- Reset Petri nets

Transformation rules

Unfolding

- basically a resolution step
- From $\frac{p(X,Y) :- Y=0, q(X)}{q(X) :- X>2, r}$
 $q(X) :- X<1, s$
- To $\frac{p(X,Y) :- Y=0, X>2, r}{p(X,Y) :- Y=0, X<1, s}$
 $q(X) :- X>2, r$
 $q(X) :- X<1, s$

Transformation rules

Constrained atomic definition

We add a new clause to the current program

– $\text{newpred}(X) \text{ :- } e(X), \text{ sat}(X, \varphi)$

where newpred is a fresh predicate symbol

Transformation rules

Constrained atomic folding

- Inverse of unfolding
- From $p(X) :- X=2, \underline{q(X)}$
 $\text{new}q(X) :- X>1, q(X)$
- To $p(X) :- X=2, \underline{\text{new}q(X)}$
 $\text{new}q(X) :- X>1, q(X)$
- Notice that $X=2$ implies $X>1$

Transformation rules

Clause removal

Remove clauses with unsatisfiable constraints

- $p(X) :- X=0, X=1.$

Remove clauses subsumed by other clauses of the form $H :- c$ where c is a constraint

- For example $q(Y) :- Y>2, p(X,Y)$
is subsumed by $q(Y) :- Y>0.$

Unfold procedure

Unfold once, then unfold as long as in the body of a clause obtained by unfolding there is an atom of one of the following forms:

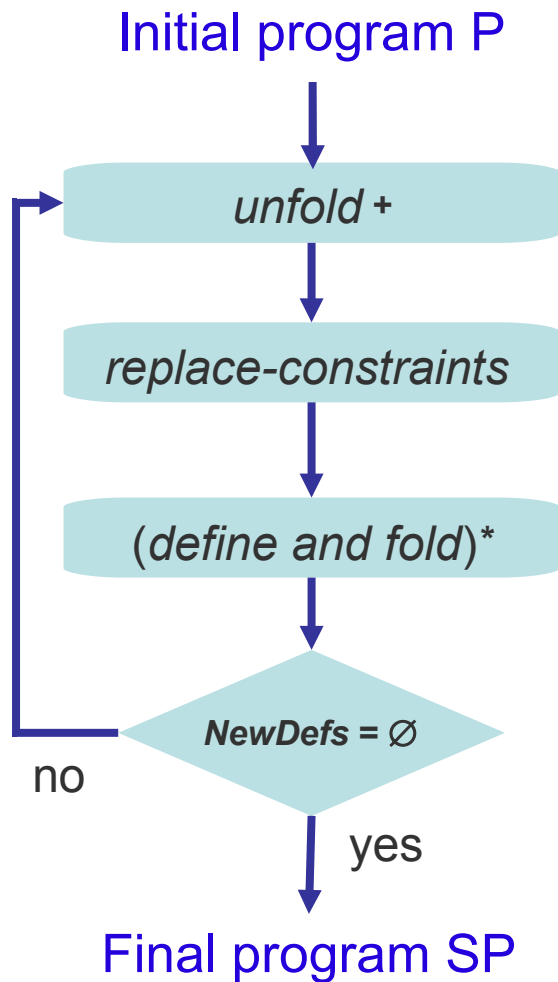
- $t(s_1, s_2), ts(s, ss)$
- $sat(s, e)$, where e is an elementary property,
- $sat(s, not(\psi)), sat(s, and(\psi_1, \psi_2)), sat(s, ex(\psi_1))$
- $sat_all(ss, \psi_1)$, where ss is a non-variable list

Clause removal

We do not repeatedly unfold atoms $sat(s, eu(\psi))$ and $sat(s, af(\psi))$

$Unfold(\gamma, \Gamma)$ terminates for any clause γ with a ground CTL formula

Specialization strategy



Constraint Generalization Techniques
(anticipate future needs for folding)