Automatic Proofs of Protocols via Program Transformation

Alberto Pettorossi
DISP - Universita' di Roma Tor Vergata
00133 Rome, Italy
pettorossi@info.uniroma2.it

Maurizio Proietti
IASI-CNR
00185 Rome, Italy
proietti@iasi.rm.cnr.it

MSRAS 2004 - Plock (Poland) June 7-9, 2004

Agents, Goals, and Protocols

- Assume many *agents* cooperating towards a *common goal*, maybe in the presence of antagonistic agents

- Need of *protocols* for making the agents cooperate

- Proof of *correctness* of the protocols

Mutual Exclusion

agent A_1 :
while *true* **do**
l_1: non-critical section;

l_3: **critical section;**

od

agent A_2 :
while *true* **do**
m_1: non-critical section;

m_3: **critical section;**

od

- ***common goal*: at most one agent in its critical section, i.e., mutual exclusion**

Peterson's Protocol

$q_1 := false;\quad s := 1;\quad q_2 := false;$

agent A_1 :
while *true* **do**
l_1: non-critical section;
l_2: $q_1 := true;\quad s := 1;$
await $(\neg q_2) \vee (s=2);$
l_3: **critical section;**
l_4: $q_1 := false;$
od

agent A_2 :
while *true* **do**
m_1: non-critical section;
m_2: $q_2 := true;\quad s := 2;$
await $(\neg q_1) \vee (s=1);$
m_3: **critical section;**
m_4: $q_2 := false;$
od

From Local Knowledge to Global Knowledge

- from low level mutual exclusion to high level mutual exclusion

  (exclusive writing of shared variables) (exclusive execution of critical sections)

- from 2 agents to n agents mutual exclusion:

$q[1..n]$ initially 0 $s[1..n-1]$ initially 1

agent A_i $(1<i<n)$:

l_2: for j=1, .., n−1 do $q[i]=j$; $s[j]=1$;

await $(\forall k.\ k\neq i \rightarrow q[k]<j) \vee (s[j]=i)$

od

l_4: $q[i]=0$

- correctness of Peterson's protocol
  - 2 agents: informal proof (10 lines)
  - n agents: no proof

Issues on Correctness

- writing of shared variables by one / more agents

- atomicity of single / compound tests

- failure / no failure of agents

Need of formalization of properties

mutual exclusion: $\neg\ (A_1 \text{ at } l_3 \ \wedge\ A_2 \text{ at } m_3)$

no starvation: $A_1 \text{ at } l_2 \ \rightarrow\ \underline{\text{after finite time}}\ A_1 \text{ at } l_3$

$q_1 := false;\quad s := 1;\quad q_2 := false;$

agent A_1 :

while *true* **do**

l_1: non-critical section;

l_2: $q_1 := true;$ $s := 1;$

await $\neg q_2 \vee s{=}2$;

l_3: **critical section;**

l_4: $q_1 := false;$

od

agent A_2 :

while *true* **do**

m_1: non-critical section;

m_2: $q_2 := true;$ $s := 2;$

await $(\neg q_1) \vee (s{=}1)$;

m_3: **critical section;**

m_4: $q_2 := false;$

od

Formal Proofs

Logic: Computation Tree Logic (CTL)
for expressing the property φ

Proof method:

- encoding the protocol K and the property φ
into a constraint logic program P_K

- program transformation to validate the property φ
as a fact which holds in the perfect model
of the program P_K

Computational Tree Logic (CTL)

CTL: a logic with

- **states and**
- **transitions between states**

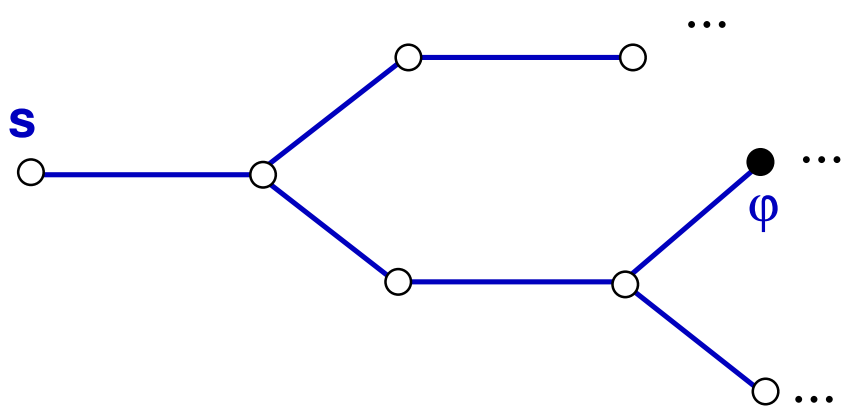
CTL formulas are built from:

- **elementary properties of states**
- **logical connectives:** $\neg \quad \wedge \quad \vee \quad \rightarrow$
- **quantifiers over paths:**
  - **E (there exists a path)**
  - **A (for all paths)**
- **quantifiers along a path:**
  - **F (there exists a state in the path / in the Future)**
  - **G (for all states in the path / Globally)**

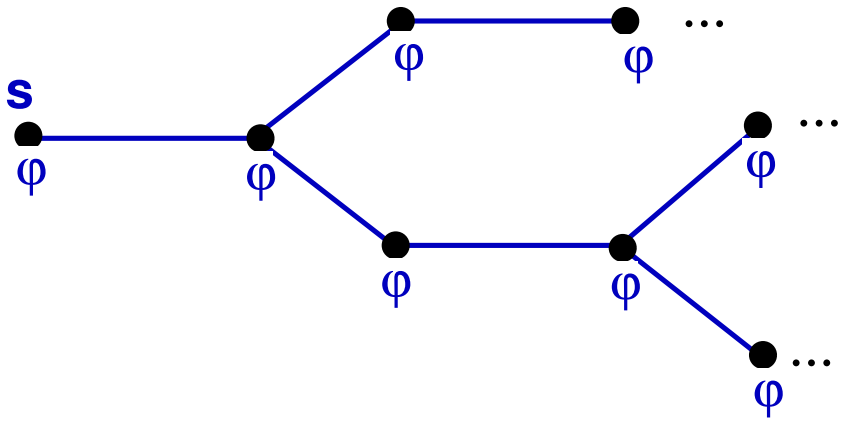
Semantics of CTL

a Kripke structure K

K,s ⊨ EF φ :



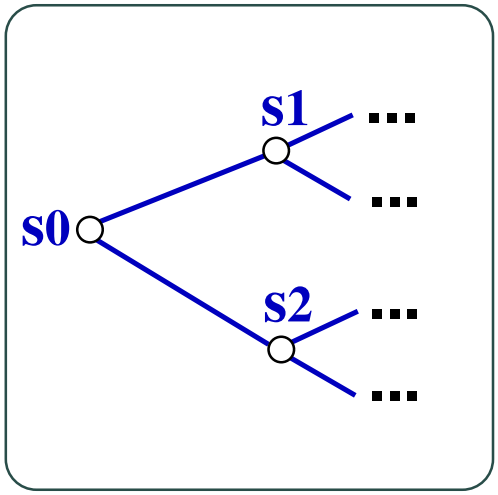
K,s ⊨ AG φ :



AG φ ↔ ¬ EF ¬ φ

Protocol and Property

Protocol K:



states as finite sets of terms
transitions as constraints on terms

Property φ:

in K at s0 property φ holds

To show $K, s0 \models \varphi$

we introduce a constraint logic program P_K and sat(s0,φ) such that

perfect model

$K, s0 \models \varphi$ iff $sat(s0,\varphi) \in M(P_K)$

The Verification Method

1. **transform** program $P_K \wedge new0 \leftarrow sat(s0,\varphi)$

   into a program Q such that the perfect model is preserved

2. **check** that $new0 \leftarrow \quad \in Q$

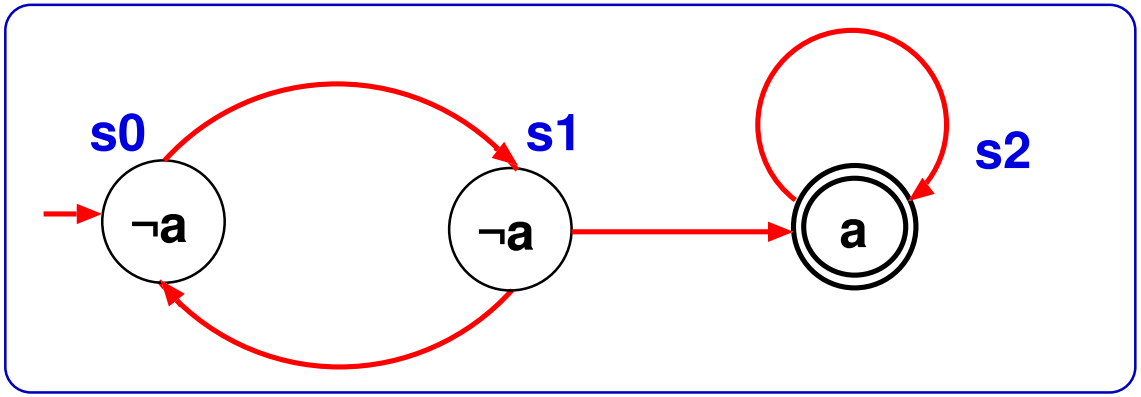
Validation of the method:

If $new0 \leftarrow \in Q$ *then* $sat(s0,\varphi) \in M(P_K)$

If $new0 \leftarrow \in Q$ *then* $K, s0 \models \varphi$

Verifying a Finite-State System

K :



To prove: $K, s0 \models AG\ EF\ a$

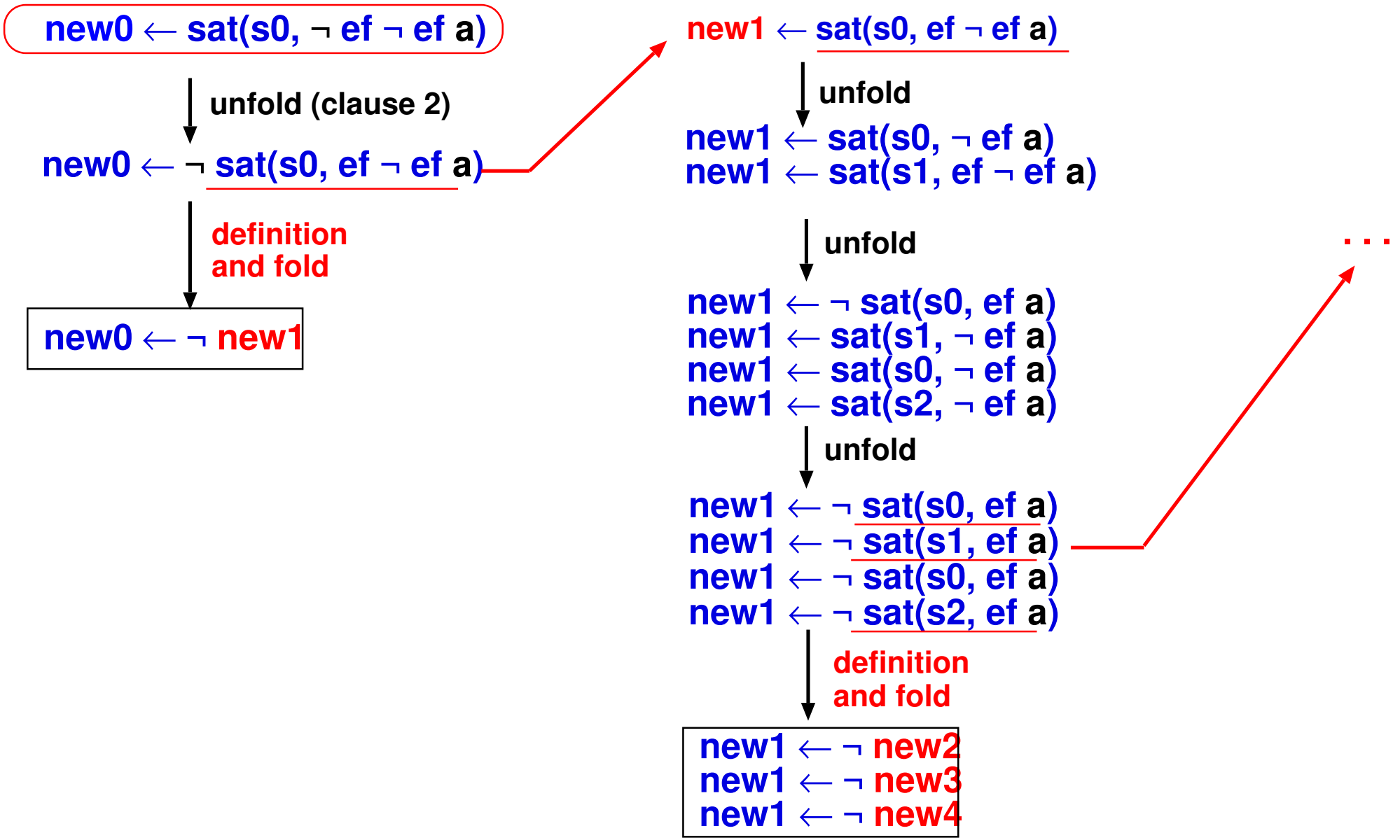
that is, $K, s0 \models \neg\ EF \neg\ EF\ a$

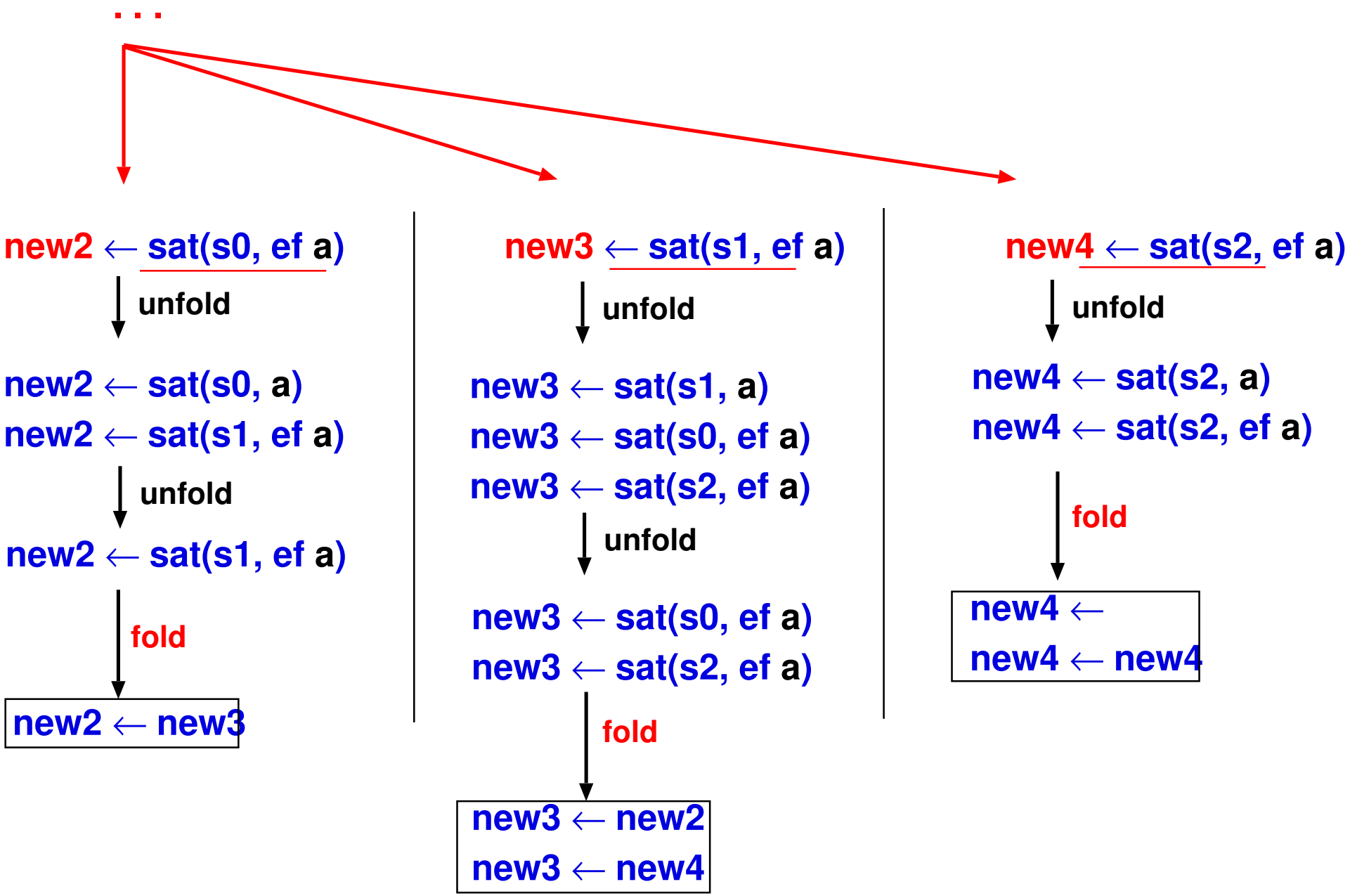
we show: $sat(s0, \neg\ ef \neg\ ef\ a) \in M(P_K)$

P_K :

1. $sat(s2, a) \leftarrow$
2. $sat(S, \neg\ F) \leftarrow \neg\ sat(S, F)$
3. $sat(S, ef\ F) \leftarrow sat(S, F)$
4. $sat(s0, ef\ F) \leftarrow sat(s1, ef\ F)$
5. $sat(s1, ef\ F) \leftarrow sat(s0, ef\ F)$
6. $sat(s1, ef\ F) \leftarrow sat(s2, ef\ F)$
7. $sat(s2, ef\ F) \leftarrow sat(s2, ef\ F)$

$new0 \leftarrow sat(s0, \neg\ ef \neg\ ef\ a)$


new0 ← sat(s0, ¬ ef ¬ ef a)
unfold (clause 2)
new0 ← ¬ sat(s0, ef ¬ ef a)
definition
and fold
new0 ← ¬ new1
new1 ← sat(s0, ef ¬ ef a)
unfold
new1 ← sat(s0, ¬ ef a)
new1 ← sat(s1, ef ¬ ef a)
unfold
new1 ← ¬ sat(s0, ef a)
new1 ← sat(s1, ¬ ef a)
new1 ← sat(s0, ¬ ef a)
new1 ← sat(s2, ¬ ef a)
unfold
new1 ← ¬ sat(s0, ef a)
new1 ← ¬ sat(s1, ef a)
new1 ← ¬ sat(s0, ef a)
new1 ← ¬ sat(s2, ef a)
definition
and fold
new1 ← ¬ new2
new1 ← ¬ new3
new1 ← ¬ new4
. . .



No new definitons need to be introduced.

Current program:

```
new0 ← ¬ new1

new1 ← ¬ new2
new1 ← ¬ new3
new1 ← ¬ new4

new2 ← new3

new3 ← new2
new3 ← new4

new4 ←
new4 ← new4
```

by unfolding and subsumption:

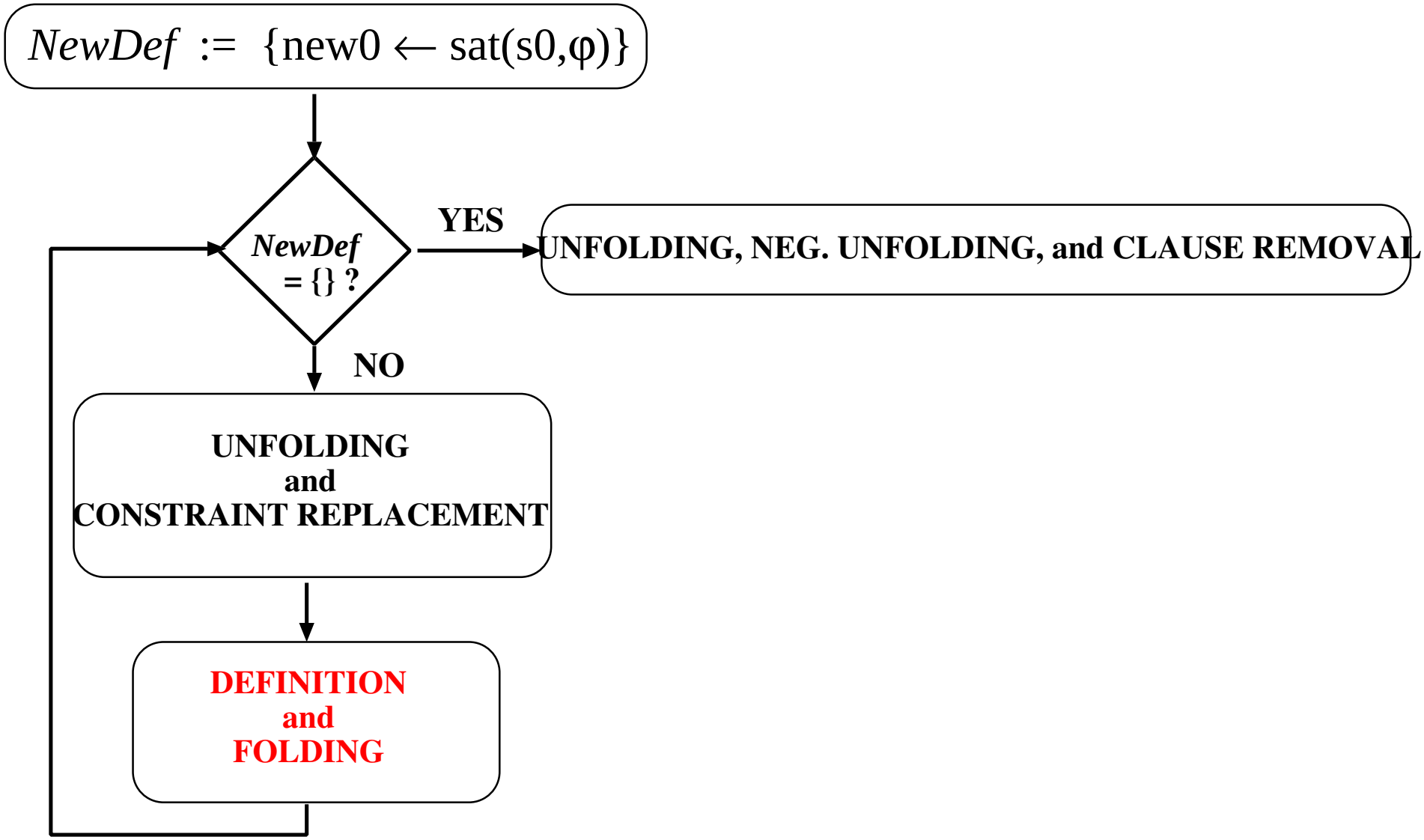
```
new0 ← ¬ new1

new1 ← ¬ new2
new1 ← ¬ new3
new1 ← ¬ new4

new2 ←

new3 ←

new4 ←
```

by negative unfolding and subsumption:


```
new0 ←

new2 ←

new3 ←

new4 ←
```

K,s0 ⊨ AG EF a holds

The Transformation Strategy



Other algorithms: Stirling+Walker, Clarke et al.

Transformation Rules

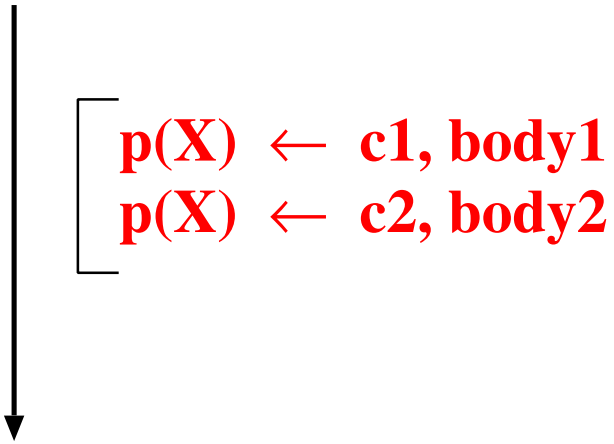
R1. Definition
R2. Unfolding
Negative Unfolding
R3. Folding
R4. Clause Removal
R5. Constraint Replacement

see also: Bensaou-Guessarian, Etalle-Gabbrielli, Maher

R2. Unfolding

Given a clause:

$$q(X) \leftarrow c, p(X), r(X,Z)$$



Replace it by the set of clauses:

$$q(X) \leftarrow c, c1, body1, r(X,Z)$$
$$q(X) \leftarrow c, c2, body2, r(X,Z)$$

and delete the clauses with unsatisfiable constraints

R2. Negative Unfolding

Given a clause :

$$H \leftarrow c, G1, \neg p(t), G2$$

1. **If no clause defines $p(X)$ then replace it by**

$$H \leftarrow c, G1, G2$$

2. **If $p(X) \leftarrow$ is in the current program then delete it**

R3. Folding

Given a clause λ of the form:

$$H \leftarrow c, G1, B, G2$$

and a definition δ of the form:

$$new(X) \leftarrow d, B$$

such that $c \rightarrow d$

Replace λ by

$$H \leftarrow c, G1, new(X), G2$$

R4. Clause Removal

- Remove clause with unsatisfiable constraints

$p(X) \leftarrow X>Y, Y>X, q(Y)$

- Remove Useless Clauses

$p \leftarrow q$

$q \leftarrow p$

$r \leftarrow$

- Remove Subsumed Clauses

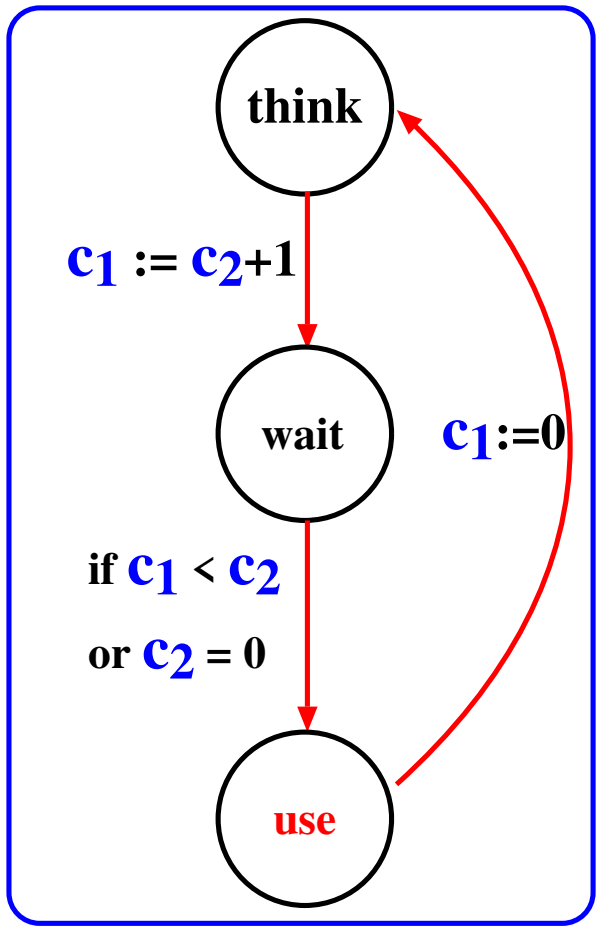
$r \leftarrow p$

$r \leftarrow$

The Bakery Protocol [Lamport 1974]

Agents A_1 and A_2 want to use a shared resource.

agent A_1 is <control state, counter> : ($\in$ {think, wait, use}; $\in$ N)



initial state: <think,0, think,0> (for A_1, for A_2)

Unlike Peterson protocol:

- each agent writes its own variables only
- the system is infinite state

Formalization of Bakery

Given K, <use,C1, use,C2> ⊨ unsafe

mutual exclusion : K, <think,0, think,0> ⊨ EF unsafe

P_K:

sat(<use,C1, use,C2>, unsafe) ←

sat(S, not(F)) ← ¬ sat(S,F)

sat(S, ef(F)) ← sat(S,F)

sat(S, ef(F)) ← trans(S,T), sat(T,ef(F))

trans(<think,A, S,B>, <wait,A1, S,B>) ← A1=B+1, A>=0, B>=0

trans(<wait,A, S,B>, <use,A, S,B>) ← A<B, A>=0

trans(<wait,A, S,B>, <use,A, S,B>) ← B=0, A>=0

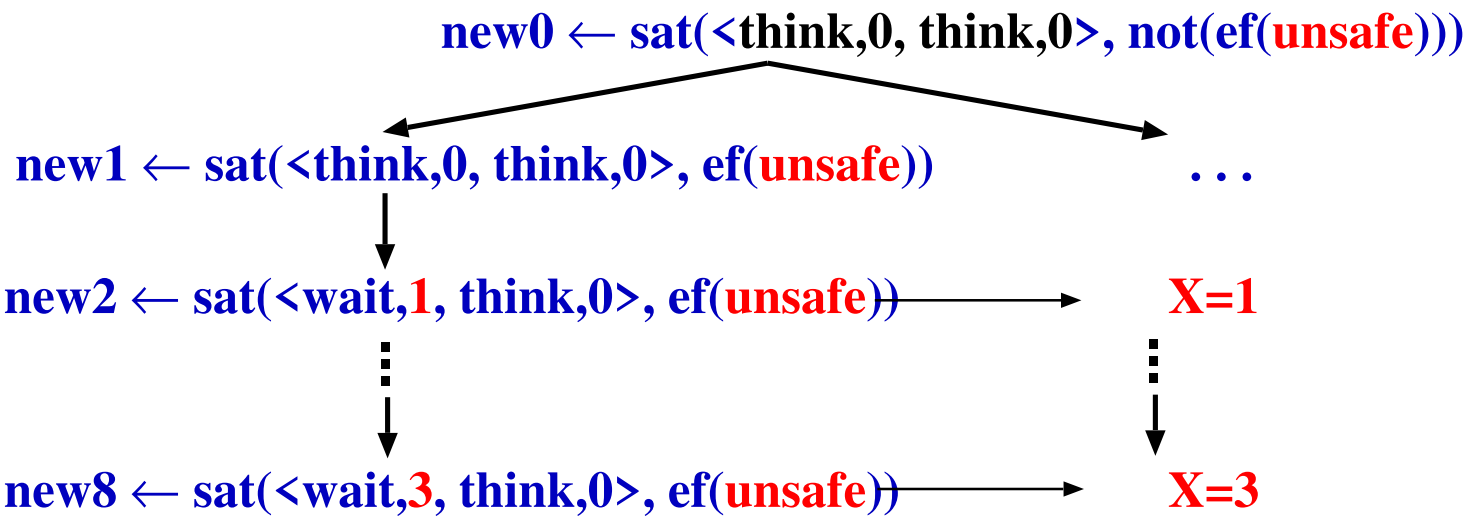
trans(<use,A, S,B>, <think,A1, S,B>) ← A1=0, A>=0, B>=0

...

new0 ← sat(<think,0, think,0>, not(ef(unsafe)))

Termination

- Termination is guaranteed for finite state systems
- Termination is not guaranteed for infinite state systems.
  Infinitely many new definitions may be introduced. TREE of DEFINITIONS:

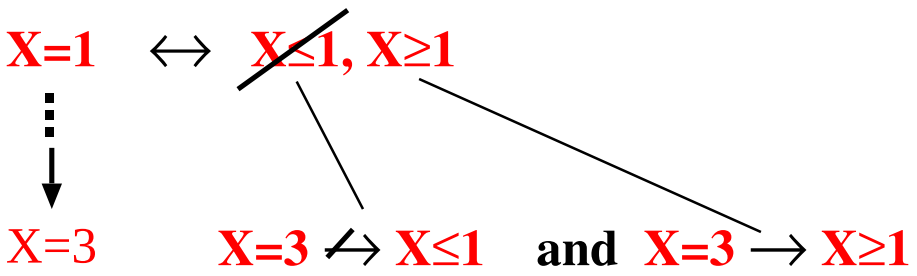


- Suitable generalizations may avoid the introduction of infinitely many definitions:

gen(X) ← X≥1, sat(<wait,X, think,0>, ef(unsafe))

Generalization

- Constraint generalization via **widening**:



- Widening avoids the introduction of an infinite number of new definitions and **guarantees the termination**

- However, some property φ of interest **may not be proved**,
  i.e., the final program Q may not contain $new0 \leftarrow$ even if $K, s0 \models \varphi$

Our method is: **sound** and **complete** for **finite** state systems
sound but **not complete** for **infinite** state systems

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**

Future work

Consider other infinite state systems

Automation issues:
- how to automatically encode systems and properties
- how to automatically generalize

Characterize the generalization strategies w.r.t. classes of properties

---

MAP transformation/verification/synthesis system available at
`http://www.iasi.rm.cnr.it/~proietti`

Related work

- K.L. McMillan et al.: reduction of infinite state system verification to finite state system verifications via abstraction (CAV 00)
- N. Shankar: use of abstraction for reducing N-process verification to 2-process verification (Concur 00)
- L. Prensa Nieto: Owicki-Gries assertional method for proving of the parametrized Ticket protocol (FMPPTA 01)
- L. Fribourg-H. Olsen: use of Presburgher Arithmetics for verifying safety properties of Petri Nets (Constraints 97)
- Y.S. Ramakrishna et al.: Model checking via tabled resolution (CAV 97
- G. Delzanno-A. Podelski: use of CLP and fixpoint approximations for verifying CTL properties of infinite state systems (TACAS 99)
- M. Leuschel-T.Massart: use of partial deduction and abstract interpretation for verifying safety properties of infinite state systems (Lopstr 99)
- A. Roychoudhury-I.V. Ramakrishnan: Automatic verification of parametrized protocols (CAV 01)