Generalization Strategies for the Verification of Infinite State Systems

Fabio Fioravanti
Dip. Scienze, University of Chieti-Pescara, Italy

joint work with

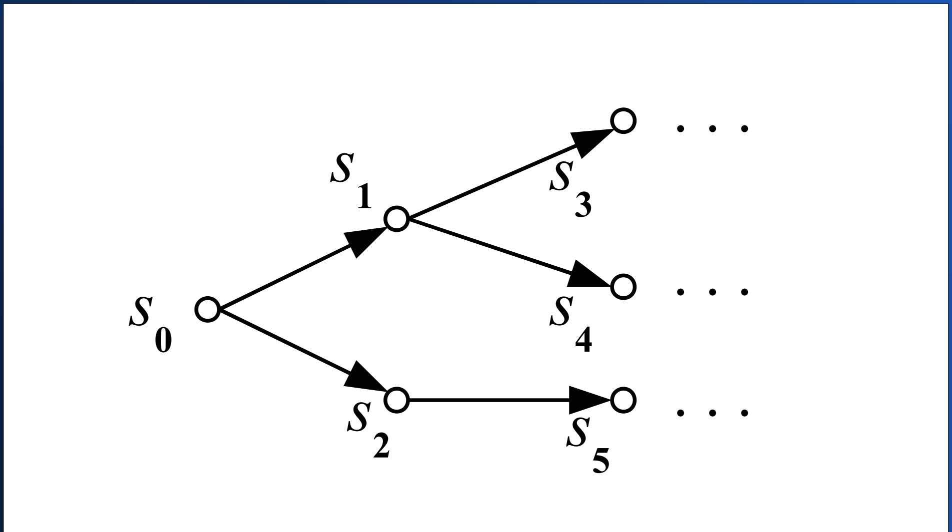
Alberto Pettorossi, Valerio Senni
DISP, University of Rome Tor Vergata, Italy
and
Maurizio Proietti
IASI-CNR, Rome, Italy

Outline

- Verification of infinite state systems
  - Computational tree logic
  - Constraint logic programming
- Two-phase Verification method
  - Rule-based program specialization
    - Generalization strategies
  - Perfect model computation
- Experimental evaluation

Infinite state systems

- The behaviour of a concurrent system can be represented as a state transition system which generates infinite computation paths:



Computational Tree Logic

- Properties are expressed in CTL, a propositional logic augmented with:
  - quantifiers over paths: E (Exists), A (All), and
  - temporal operators along paths: 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 in polynomial time for finite state systems
  - undecidable for infinite state systems

Computational Tree Logic

Let K be a Kripke structure (S,I,R,L), s a state, and Elem a set of el. prop. where S set of states, $I \subseteq S$ initial states, $R \subseteq SxS$ transition relation, $L: S \rightarrow P(Elem)$ labeling function.

Let π be an infinite list $[s_0, \ldots, s_k, \ldots]$ of states and **d**, ϕ, ψ be CTL formulas

$\mathrm{K}, s \models \mathbf{d}$ iff $\mathbf{d} \in L(s)$

$\mathrm{K}, s \models \neg \phi$ iff $\mathrm{K}, s \models \phi$ does not hold

$\mathrm{K}, s \models \phi \wedge \psi$ iff $\mathrm{K}, s \models \phi$ and $\mathrm{K}, s \models \psi$

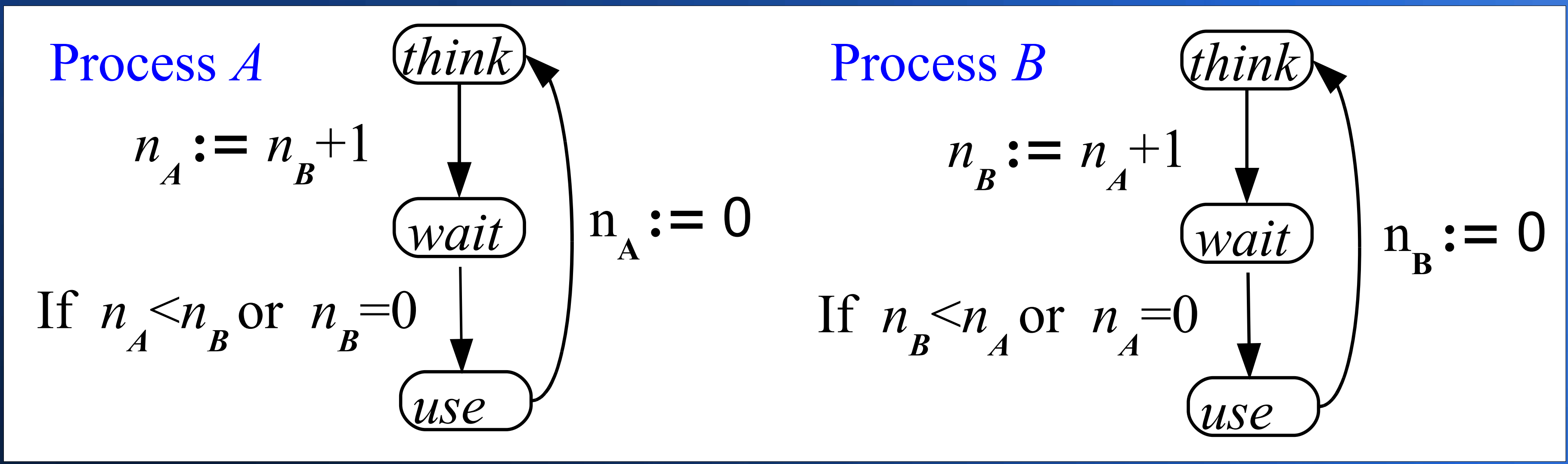
$\mathrm{K}, s \models \mathbf{EX}\, \phi$ iff $\exists\, \pi = [s_0, s_1, \ldots]$, $s = s_0$, and $\mathrm{K}, s_1 \models \phi$

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

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

The Bakery Protocol (Lamport)

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



System: A || B
Path: <*think*,0,*think*,0>→ <*wait*,1,*think*,0> →<*wait*,1,*wait*,2>→ <*use*,1,*wait*,2>→ ---

Mutual Exclusion: <*think*,0,*think*,0> |= ¬ *EF unsafe*
where, for all n_A, n_B: <*use*,n_A,*use*,n_B> |= *unsafe*

Temporal Properties as Constraint Logic Programs

A system S and the temporal logic are encoded by a CLP program.

- the transition relation is encoded by a binary predicate tr, like f.e.:

```
tr(<think,A,S,B>,<wait,A1,S,B>) :- A1=B+1.
tr(<wait,A,S,B>,<use,A,S,B>) :- A<B.
tr(<wait,A,S,B>,<use,A,S,B>) :- B=0.
tr(<use,A,S,B>,<think,A1,S,B>) :- A1=0.
```

+ similar clauses for process B

- the initial states: initial(<think, A, think, B>) :- A=0, B=0.
- the elementary properties: elem(<use,A,use,B>,unsafe).

Temporal Properties as Constraint Logic Programs

The satisfaction relation |= is encoded by a binary predicate sat

```
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(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)
```

where ts(X,Ys) holds iff Ys is a list of all the successor states of X

Temporal Properties as Constraint Logic Programs

The property to be verified is defined by a predicate prop.

s.t. prop $\equiv_{def}$ $\forall X(\text{initial}(X) \rightarrow \text{sat}(X,\varphi))$

$\neg \exists X(\text{initial}(X) \wedge \neg \text{sat}(X,\varphi))$

encoded as follows

```
g1 : prop :- \+ negprop
g2 : negprop :- initial(X), \+ sat(X,φ)
```

Correctness of the Encoding

Let P_s be the set of clauses defining predicates sat, tr, ts, sat_all, prop, negprop. P_s 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,

$$(\text{for all states } s \in I,\ K,s \models \varphi) \quad \text{iff} \quad prop \in M(P_s).$$

But...

- Bottom-up construction of $M(P_s)$ from facts may not terminate because $M(P_s)$ is infinite.
- Top-down evaluation of P_s from *prop* may not terminate due to infinite computation paths.

Two-phase Verification Method

- Phase 1: specialize P_s w.r.t. the query *prop:*

  $$P_s \rightarrow \cdots \rightarrow SpP_s \quad \text{s.t.} \quad \text{prop} \in M(P_s) \text{ iff prop} \in M(SpP_s)$$

  and keep only the clauses on which the predicate prop depends. SpP_s 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 $M(SpP_s)$ (may not terminate)

Specialization strategy

- Input: The program P_s
  Output : A stratified program SpP_s such that
  prop $\in$ M(P_s) iff prop $\in$ M(SpP_s).
- *SpPs* := {g1}; InDefs := {g2}; Defs := {};
- while (there exists a clause γ in InDefs) do
  - Unfold(γ,Γ);
  - Generalize&Fold(Defs, Γ, NewDefs, Φ);
  - *SpPs* := *SpPs* $\cup$ Φ; InDefs := (InDefs − {γ}) $\cup$ 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

- For limiting the number of clauses introduced by definition, sometimes we introduce definitions containing a generalized constraint
- Well quasi orderings: generalization is eventually applied
- Generalization operators: each definition can be generalized a finite number of times only
- Selecting a good generalization strategy is not trivial
  - Too coarse -> unable to prove property
  - Too fine-grained -> high verification times

The constraint domain Lin_k

- Lin_k are linear inequations over k distinct variables $X_1,...,X_k$
- Constraints of Lin_k are conjunctions of atomic constraints of the form
  - $p \leq 0$ or $p < 0$

  where p is a polynomial of the form
  - $q_0 + q_1X_1 + ... + q_kX_k$

  and q_i 's are integers

Well-quasi orderings

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

  for every infinite sequence $e_0, e_1, \ldots$ of elements of S, there exist i and j such that $i < j$ and $e_i \preceq e_j$.

HomeoCoeff wqo

- HomeoCoeff compares sequences of absolute values of integer coefficients occurring in polynomials

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

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

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

- SumCoeff compares the sum of the absolute value of coefficients occurring in polynomials

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

Generalization operators

- Given a wqo $\preceq$, the generalization of a constraint c w.r.t. a constraint d is a constraint $c \ominus d$ such that
  - $d \sqsubseteq c \ominus d$
  - $c \ominus d \preceq c$
- $c \ominus d$ can replace d in a candidate definition for folding
- 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

Let $c = a_1,...,a_m$ and $d = b_1,...,b_n$

- Top: $c \ominus d$ is the constraint *true*
- Widen: $c \ominus d$ is the conjunction of all a_i's such that $d \sqsubseteq a_i$
- WidenPlus: $c \ominus d$ is the conjunction of all a_i's such that $d \sqsubseteq a_i$ and of all b_j's such that $b_j \preceq c$
- CHWiden and CHWidenPlus 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:
  - bakery2 (safety and liveness)
  - bakery3 (safety)
  - Mutast (safety)
  - Peterson (safety for N processes)
  - Ticket (safety and liveness)

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

Generalization G: EXAMPLE wqo W:	*CHWiden* HC	*CHWiden* SC	*CHWidenPlus* SC	*Top* HC	*Top* SC	*Widen* HC	*Widen* SC	*WidenPlus* MC	*WidenPlus* SC
Bakery 2 (safety)	20	70	20	30	40	20	60	30	20
	20	50	20	20	30	20	40	30	20
Bakery 2 (liveness)	60	120	80	80	100	70	130	80	70
	40	80	60	50	60	50	90	60	50
Bakery 3 (safety)	160	800	180	2420	3010	170	750	180	160
	150	430	170	730	680	160	380	170	150
MutAst	230	440	440	2870	2490	220	370	70	140
	200	390	420	330	220	190	320	70	140
Peterson N	∞	∞	1370	∞	∞	∞	∞	210	230
	∞	410	1370	∞	30	∞	250	210	230
Ticket (safety)	30	30	20	20	30	20	30	20	40
	30	20	10	20	20	20	20	10	30
Ticket (liveness)	90	120	120	100	110	100	100	110	110
	50	70	70	60	60	60	50	60	60
Berkeley RISC	60	60	200	70	30	50	50	30	30
	60	40	170	50	20	50	30	30	30
DEC Firefly	190	120	340	100	80	180	120	30	20
	100	60	160	40	20	90	60	30	20
IEEE Futurebus+	∞	47260	47260	∞	15630	∞	4720	100	2460
	∞	290	290	∞	30	∞	230	100	270
Illinois University	50	80	40	140	90	50	70	40	20
	50	60	40	60	30	50	50	30	10
MESI	100	50	130	100	70	100	50	30	30
	80	40	120	50	20	80	40	30	30
MOESI	980	160	180	930	100	940	160	50	60
	950	60	80	860	30	910	60	50	50
Synapse N+1	30	10	10	20	20	20	10	10	10
	20	10	10	20	10	10	10	10	10
Xerox PARC Dragon	1230	80	280	1140	50	1210	70	30	40
	1180	60	260	1110	20	1160	50	30	40
Barber	41380	30150	2740	∞	∞	40750	29030	1210	1170
	3260	3100	2620	900	410	2630	1620	1170	1130
Bounded Buffer	73990	370	6790	71870	20	75330	340	3520	3540
	73190	170	6780	71850	20	74550	140	2040	2060
Unbounded Buffer	∞	∞	410	∞	∞	∞	∞	3890	3890
	310	130	410	140	10	280	100	360	360
CSM	∞	∞	4710	∞	∞	∞	∞	6380	6580
	∞	620	4700	30	20	∞	440	6300	6300
Insertion Sort	80	80	160	110	80	70	70	90	100
	80	60	150	30	20	70	50	90	100
Selection Sort	∞	∞	200	∞	∞	∞	∞	∞	190
	380	80	200	40	40	340	70	770	180
Office Light Control	40	50	50	40	30	50	50	50	50
	30	40	40	30	30	40	40	40	40
Reset Petri Nets	∞	∞	∞	∞	∞	∞	∞	0	0
	10	10	10	0	10	0	10	0	0

Analysis

- Precision (number of properties proved) and average verification time
  - SumCoeff &WidenPlus 23/23 (820 ms)
  - MaxCoeff &WidenPlus 22/23 (730 ms)
  - SumCoeff &CHWidenPlus 22/23 (2990 ms)
- Top and Widen are fast but not accurate
  - information about the call can be lost

Comparison with other systems

- Action Language Verifier (Bultan 01)
  - combines BDD-based symbolic manipulation for boolean and enumerated types, with a solver for linear constraints on integers
- DMC (Delzanno 01)
  - computes (approximated) least and greatest models of CLP(R) programs
- HyTech (Henzinger 97)
  - model checker for hybrid systems

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

Analysis

- Precision (number of properties proved) and average verification time
  - MAP 23/23 (820 ms)
  - DMC (with abstraction) 19/23 (820 ms)
  - ALV (default option) 18/23 (8480 ms)
  - HyTech (backwards) 17/23 (70 ms)

Analysis

- Bounded and Unbounded Buffer can be easily verified by backward reachability
  - The specialization phase is redundant
  - MAP slower than other systems
- Peterson 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

Transformation rules

- Unfolding
  - basically a resolution step
  - From p(X,Y) :- Y=0, q(X)
q(X) :- X>2, r
q(X) :- X<1, s
  - To 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
  - newpred(X) :- e(X), sat(X,φ)

where newpred is a fresh predicate symbol

Transformation rules

- Constrained atomic folding
  - Inverse of unfolding
  - From p(X) :- X=2, q(X)
    newq(X) :- X>1, q(X)
  - To p(X) :- X=2, newq(X)
    newq(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 contraint
  - 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(s1,s2), ts(s,ss)
  - sat(s,e), where e is an elementary property,
  - sat(s,not(ψ)), sat(s,and(ψ1,ψ2)),sat(s,ex(ψ1))
  - sat_all(ss,ψ1), where ss is a non-variable list
- Clause removal
- We do not repeatedly unfold atoms sat(s,eu(ψ)) and sat(s,af(ψ))
- Unfold(γ,Γ) terminates for any clause γ with a ground CTL formula