Automated Verification of Multithreaded and Mobile Programs
via
Infinite-state Symbolic Model Checking

Giorgio Delzanno

D.I.S.I. - Dipartimento di Informatica e Scienze dell'Informazione

University of Genova

Grenoble, March 2003

Background

- Practical examples of multithreaded programs and protocols for distributed systems often have
  - *unbounded data*: generation of fresh names, ...
  - *unbounded control*: spawning of new processes, ...
  - *unbounded data and control*: multithreaded software
  - *process mobility*: dynamic reconfiguration of the network programs,...

- Can we still apply *automated verification techniques* when their *state-space* becomes infinite in *one* or *more dimensions*?

Bounded control, unbounded data

Constraints to symbolically represent data

- Henzinger-Ho-Wong-Toi. *HyTech: a Model Checker for Hybrid Systems*, CAV'97 BASED ON THE POLYHEDRA library
- Bultan-Gerber-Pugh. *Symbolic Model Checking of Infinite State Systems Using Presburger Arithmetics*, CAV'97 BASED ON THE OMEGA LIBRARY
- ...

Unbounded control, bounded data

Constraints to symbolically represent sets of processes

- Browne-Clarke-Grumberg. *Reasoning about Networks with Many Identical Finite State Processes*, IC 1989
- Bouajjani-Jonsson-Nilsson-Touili. *Regular Model Checking*, CAV 00

BASED ON REGULAR LANGUAGES

- Esparza-Finkel-Mayr. *Verification of Broadcast Protocols*, LICS 99
- Delzanno-Esparza-Podelski. *Constraint-based Verification of Broadcast Protocols*, CSL 99
- Delzanno-Raskin-Van Begin. *Towards Verif. of Multithreaded Progranms*, TACAS 02

SYMBOLIC ANALYSIS FOR PETRI NETS

Unbounded data and parameterized control

- Abdulla-Jonsson. *Verifying Networks of Timed Processes*, TACAS'98
- Abdulla-Nylén. *Better is Better than Well: On Efficient Verification of Infinite-State Systems*, LICS'00

Based on symbolic model checking

- Arons-Pnueli-Ruah-Xu-Zuck. *Parameterized Verification with Automatically Computed Inductive Assertions*, CAV'01

Based on abstractions+deductive verification

Current Research Line

Overall goal

To develop *sound* and *fully-automatic* methods based on *constraint programming* technology for the verification of concurrent systems with

- *unbounded control*
- *unbounded data*
- *process mobility*

Practical applications

Consistency protocols for distributed systems with shared memory

Cache coherence protocols for multi-processors and multi-line caches

Security protocols

...

Abstractions of *multithreaded* programs

TDL: Thread Definition Language

A language for concurrent systems based on CFSMs enriched with

- local variables over an infinite *name domain*
- transitions of the form $s \stackrel{\alpha}{\longrightarrow} s'[\varphi]$ where
  - s and s' are control locations,
  - φ contains guards ($x = y$, $x \neq y$) and assignments over local variables and message templates,
  - α is a channel expression

TDL: Thread Definition Language

- A primitive for generations of new names $x := new$ where x is a local variable
- A primite *run* T *with* α for spawning a new thread T with initialization of local variables α
- Rendez-vous communication: $e!m$, $e?m$ where
  - e (channel) is either a constant c or a local variable x
  - m (message) is a tuple $\langle x_1, \ldots, x_n \rangle$ of (local) variables
- Variables (ranging over an infinite domain of names) are used as ports to achieve *process mobility*

A Challenge-Response Protocol

$Thread\ Alice(local\ id_A, n_A, m_A);$

$$init_A \xrightarrow{fresh} gen_A[n_A := new]$$

$$gen_A \xrightarrow{c!\langle n_A \rangle} wait_A[true]$$

$$wait_A \xrightarrow{n_A?\langle y \rangle} stop_A[m_A := y]$$

$Thread\ Bob(local\ id_B, n_B, m_B);$

$$init_B \xrightarrow{c?\langle x \rangle} gen_B[n_B := x]$$

$$gen_B \xrightarrow{fresh} ready_B[m_B := new]$$

$$ready_B \xrightarrow{n_B!\langle m_B \rangle} stop_B[true]$$

Initiator

$Thread\ Main;$

$Local\ x;$

$$init_M \xrightarrow{id} create[x := new]$$

$$create \xrightarrow{new_A} init_M[run\ Alice\ with\ id_A := x, n_A := \bot, m_A := \bot, x := \bot]$$

$$create \xrightarrow{new_B} init_M[run\ Bob\ with\ id_B := x, n_B := \bot, m_B := \bot, x := \bot]$$

Sample Run

Global configuration

$$\langle \underbrace{\langle i_1, \ldots, i_n \rangle}_{used\ names}, \underbrace{\ell_1, \ldots, \ell_k}_{local\ states} \rangle$$

Run

$$\langle \{\bot, i_1, i_2\}, \langle init_M, \bot \rangle, \langle init_A, i_1, \bot, \bot \rangle, \langle init_B, i_2, \bot, \bot \rangle \rangle$$

$$\Rightarrow \langle \{\bot, i_1, i_2\}, \langle init_M, \bot \rangle, \langle gen_A, i_1, a^1, \bot \rangle, \langle init_B, i_2, \bot, \bot \rangle \rangle$$

$$\Rightarrow \langle \{\bot, i_1, i_2, a^1\}, \langle init_M, \bot \rangle, \langle wait_A, i_1, a^1, \bot \rangle, \langle gen_B, i_2, a^1, \bot \rangle \rangle$$

$$\Rightarrow \langle \{\bot, i_1, i_2, a^1, a^2\}, \langle init_M, \bot \rangle, \langle wait_A, i_1, a^1, \bot \rangle, \langle ready_B, i_2, a^1, a^2 \rangle \rangle$$

$$\Rightarrow \langle \{\bot, i_1, i_2, a^1, a^2\}, \langle init_M, \bot \rangle, \langle stop_A, i_1, a^1, a^2 \rangle, \langle stop_B, i_2, a^1, a^2 \rangle \rangle$$

The Verification of our Example is Challenging

- Suppose we want to prove that at the end of every session any two agents who started the protocol eventually get to know both nonces they exchanged
- This is a verification problem for a *parameterized system* in which *individual components* have *infinitely many* possible states (we generate fresh names and new threads)

Several Problems to Solve

- We need a specification language for *parameterized systems with unbounded local data*
- We need an *assertional language* to specify safety properties
- We need *sound* and *fully automatic* procedures to validate the specification against the desired property

Low Level Specification Language

Multiset Rewriting + Constraints

- *Multiset rewriting over first order atomic formulas* (MSR) can be used as a flexible specification language for concurrent systems

- MSR has been introduced to specify *security protocols*
  - Locality of process definitions and communication via rendez-vous
  - First order terms as color for processes

- The combination of MSR with a *constraint system* $\mathcal{C}$ can be used to *symbolically* represent systems with heterogeneous data structures

- The resulting specification language is called MSR($\mathcal{C}$)

MSR(>, =) specification of the sample protocol

We use a global counter to manage fresh name generation

$$init \longrightarrow fresh(x) \mid init_M(y) \; : \; x > 0, y = 0.$$

$$fresh(x) \mid init_M(y) \longrightarrow fresh(x') \mid create(y') \; : \; x' > y', y' > x.$$

$$create(x) \longrightarrow init_M(x') \mid init_A(id', n', m') \; : \; x' = x, id' = x, n' = 0, m' = 0.$$

$$create(x) \longrightarrow init_M(x') \mid init_B(id', n', m') \; : \; x' = x, id' = x, n' = 0, m' = 0.$$

Core Protocol

$$init_A(id,n,m)|\ fresh(u) \longrightarrow gen_A(id',n',m')\ |\ fresh(u')\ :$$
$$u' > n', n' > u, m' = m, id' = id.$$

$$gen_A(id_1,n,m)|\ init_B(id_2,u,v) \longrightarrow wait_A(id_1',n',m')\ |\ gen_B(id_2',u',v')\ :$$
$$n' = n, m' = m, u' = n, v' = v, id_1' = id_1, id_2' = id_2$$

$$gen_B(id,n,m)|\ fresh(u) \longrightarrow ready_B(id',n',m')\ |\ fresh(u')\ :$$
$$u' > m', m' > u, n' = n, id' = id.$$

$$wait_A(id_1,n,m)|\ ready_B(id_2,u,v) \longrightarrow stop_A(id_1',n',m')\ |\ stop_B(id_2',u',v')\ :$$
$$n = u, n' = n, m' = v, u' = u, v' = v, id_1' = id_1, id_2' = id_2.$$

$$stop_A(id,n,m) \longrightarrow init_A(id',n',m')\ :\ n' = 0, m' = 0, id' = id.$$

$$stop_B(id,n,m) \longrightarrow init_B(id',n',m')\ :\ n' = 0, m' = 0, id' = id.$$

Configuration and Run

A Configuration is a multiset $\mathcal{M}$ of *ground* atomic formulas

One Step Rewriting

$$\underline{init_A(i,j,l)}|init_B(a,b,c)|init_A(r,s,t)|\ \underline{fresh(k)} \quad \Rightarrow$$
$$\underline{\underline{gen_A(i,j',l)}}\ |init_B(a,b,c)|init_A(r,s,t)|\ \underline{\underline{fresh(k')}}$$

using the instance rule

$$init_A(i,j,l)|fresh(k) \longrightarrow gen_A(i,j',l) \mid fresh(k')\, with\, k' > j' > k$$

Reachability $\mathcal{M}$ is reachable if $init \overset{*}{\Rightarrow} \mathcal{M}$

Properties and Assertional Language

Parameterized Verification

- Let S be the set of *good configurations.* The corresponding *safety property* holds if for any $\mathcal{M}$

$$if\ init \stackrel{*}{\Rightarrow} \mathcal{M}\ \ then\ \ \mathcal{M} \in S$$

- Dually, let U be the set of *bad configurations*, then the property holds if

$$init \notin Pre^*(U)$$

where $Pre^*(U) \ = \ \{ \ \mathcal{M} \mid \mathcal{M} \stackrel{*}{\Rightarrow} \mathcal{M}',\ \mathcal{M}' \in U \ \}$

- We have to explore a potentially infinite number of configurations

Symbolic Representation of Configurations

- Unsafe States can be represented as the *constrained configuration*:

$$stop_A(i_1, n_1, m_1) \mid stop_B(i_2, n_2, m_2) \; : \; n_1 = n_2, m_1 > m_2.$$
$$stop_A(i_3, n_3, m_3) \mid stop_B(i_4, n_4, m_4) \; : \; n_3 = n_4, m_4 > m_3.$$

- if we consider its *upward-closed denotations*

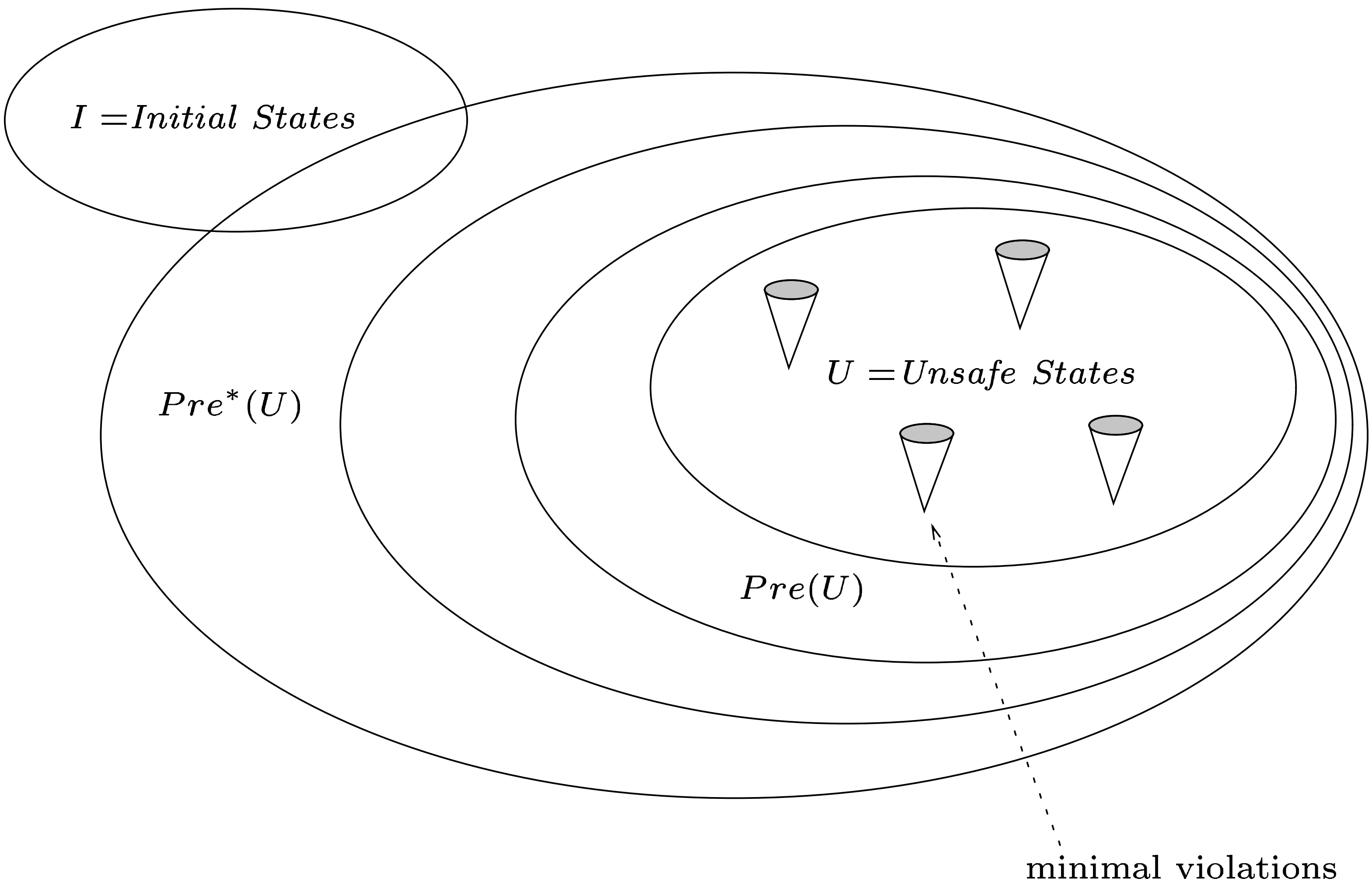
$$\begin{aligned}[\![\mathbf{U}]\!] \; = \{ \; & stop_A(i, n, m) \mid stop_B(j, n, m') \oplus \mathcal{M} \\ & \forall \; i, \; j, \; n, \;\; m \neq m', \\ & \forall \; configuration \; \mathcal{M} \; \}\end{aligned}$$

- defined in general as follows

$$[\![\mathcal{M} : \varphi]\!] \; = \{ \; \mathcal{N} \mid \sigma(\mathcal{M}) \preccurlyeq \mathcal{N}, \; \sigma \; solution \; of \; \varphi \; \}$$

Verification Procedures

Backward Reachability



Pre-image Computation

From

$$p(u) \mid \underline{\underline{p(v)}} \; : \; true$$

using the rule

$$\underline{\underline{w(x)}} \mid \underline{t(y)} \;\rightarrow\; \underline{\underline{p(x')}} \mid \underline{t(y')} : x = y, x' = x, y' = y$$

we get

$$p(u) \mid \underline{\underline{w(x)}} \mid \underline{t(y)} : x = y$$

but also

$$p(u) \mid p(v) \mid \underline{\underline{w(x)}} \mid \underline{t(y)} : x = y$$

Entailment

- We define an ordering based on *AC unification* and on the *entailment* relation of the underlying constraints:
- For instance

$$p(x, y) \mid q(z) \mid r(u) : x > y, y = z$$

$$\textit{entails}$$

$$q(z') \mid p(x', y') \ \ : x' > y'$$

- Infact,

$p(x, y) \mid q(z)$ and $q(z') \mid p(x', y')$ unify via $x = x', y = y', z = z'$

$x' > y', x' = z'$ entails $x' > y'$.

Enforcing Termination

Invariant Strengthening

- We observe that

$$\left.\begin{array}{l} [\![\mathbf{U}]\!] \subseteq [\![\mathbf{U}']\!] \\ init \notin \mathbf{Pre}^*(\mathbf{U}') \end{array}\right\} \text{ implies } \quad init \notin \mathbf{Pre}^*(\mathbf{U})$$

- This idea can be viewed as a *static widening*

Widening

- We can apply abstractions working on the components of the symbolic representation

$$\alpha(\mathcal{M} : \varphi) = \alpha_f(\alpha_m(\mathcal{M}) : \alpha_c(\varphi))$$

and extend it to **Pre** as follows $\mathbf{Pre}_\alpha(S) = \alpha(\mathbf{Pre}(S))$

Sufficient Conditions for Guarantee of Termination

- Monadic constrained configurations with constraints like $x = y$ and $x > y$

$$p(x) \mid q(y) \ : \ x > y$$

- Constrained configurations whose constraints are *separable* with respect to positions in atomic formulas

$$p(x, y) \mid q(u, z) \ : \ x = u, y > z$$

Towards Security Protocols

- The combination of constraints and uninterpreted function symbols can be used to naturally encode several protocols used for security

$$fresh(nonce(x)) \longrightarrow$$
$$fresh(nonce(y)) \mid step1(pk(a), \langle n, pk(b)\rangle) \mid net(enc(pk(b), \langle nc(n), pk(a)\rangle)) :$$
$$a > 0, b > 0, y > n, n > x$$

- We can extend the symbolic verification procedure to the new class of specifications
  - Contrary to forward exploration, in the symbolic backward approach it is not necessary to generate new constants

Conclusions

- *Push-button* verification method for infinite-state concurrent systems based on the paradigm of symbolic model checking and constraints
- Potential application to *nominal process calculi* with *unbounded control*, *fresh name generation*, and *name mobility*
- Potential application to *verification* of *security protocols*
- Specialized data structures are needed to scale up
- Abstractions/accelerations are needed for terminations (class of widening operators for security protocols?)