Control-flow in Software Model Checking

An Automata-Theoretic Approach

Javier Esparza

Institute for Formal Methods in Computer Science

University of Stuttgart

Sources of infinity

Data manipulation: unbounded counters, integer variables, lists, queues . . .

Real-time: discrete or dense domains

Parameters: number of processes, of principals, of input gates, delays, . . .

Sources of infinity

Data manipulation: unbounded counters, integer variables, lists, queues . . .

Real-time: discrete or dense domains

Parameters: number of processes, of principals, of input gates, delays, . . .

Control structures: procedures, process creation . . .

Quicksort

```
void qs (int left,int right) {
  int lo,hi,piv;
  if (left >= right) return;
  piv = a[right]; lo = left; hi = right;
  while (lo <= hi) {
    if (a[hi] > piv) {
      hi--;
    } else {
      swap a[lo],a[hi];
      lo++;
    }
  }
  qs(left,hi);
  qs(lo,right);
}
```

In this talk

Formal models for control structures.

Automata theoretic techniques to check properties.

Applications to:

- dataflow analysis,
- program correctness,
- program security,
- analysis of authorization schemes,
- ...

A control-flow hierarchy

	"plain programs"	while programs, if-goto programs
+	(possibly recursive) procedures	sequential programs
+	thread creation	synchronization-free multithread programs
+	thread synchronization	multithread programs

A Model of Sequential Programs

Programs determined by

control flow of procedures

- assignments, conditionals, loops
- procedure calls with parameter passing / return values

local variables of each procedure

global variables

State space determined by

program pointer

values of global variables

values of local variables (of current procedure)

activation records (return addresses, copies of locals)

Pushdown systems

A pushdown system (PDS) is a triple (P, Γ, Δ), where

- P is a finite set of control locations
- Γ is a finite stack alphabet
- $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of rules.

A configuration is a pair $\langle p, v \rangle$, where $p \in P$, $v \in \Gamma^*$

If $\langle p, \gamma \rangle \hookrightarrow \langle p', v \rangle \in \Delta$ then $\langle p, \gamma w \rangle \longrightarrow \langle p', vw \rangle$ for every $w \in \Gamma^*$

Normalisation: $|v| \leq 2$

Interpretation of $\langle p, \gamma v \rangle$

p holds values of global variables

γ holds (program pointer, values of local variables)

v holds stack of (return address, saved locals)

Restriction: finite datatypes

Correspondence between statements and rules

$\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle$	simple statement
$\langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \gamma'' \rangle$	procedure call
$\langle p, \gamma \rangle \hookrightarrow \langle p', \epsilon \rangle$	return statement

Reachability in pushdown systems

Key problem: compute (possibly infinite) sets of predecessors and successors

Sets I and F derived from model-checking problems are regular:
for every control point p, the set $\{w \in \Gamma^* \mid \langle p, w \rangle \in C\}$ is regular

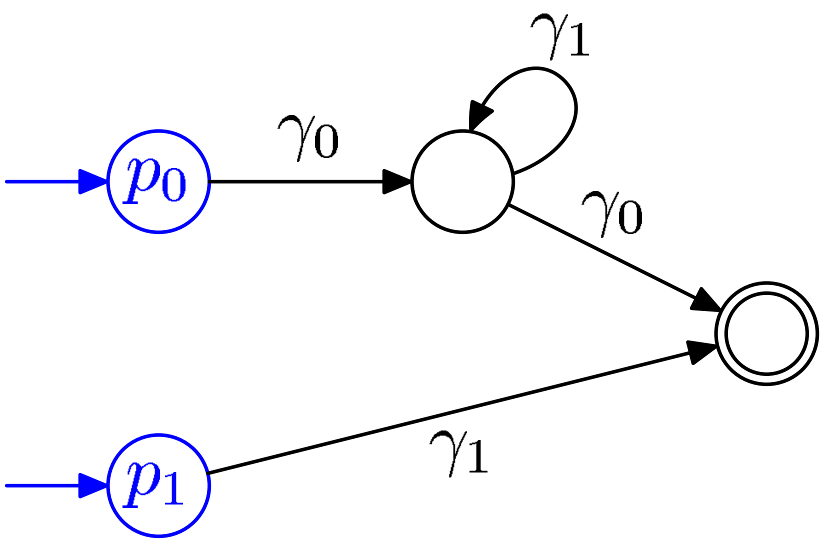
Symbolic representation: use multi-automata to represent regular sets of configurations

P as set of initial states and Γ as alphabet

$\langle p, v \rangle$ recognized if $p \xrightarrow{v} q$ for some final state q

Example: $P = \{p_0, p_1\}$ and $\Gamma = \{\gamma_0, \gamma_1\}$

Automaton coding the set $\langle p_0, \gamma_0\gamma_1^*\gamma_0\rangle \cup \langle p_1, \gamma_1\rangle$:



The key results

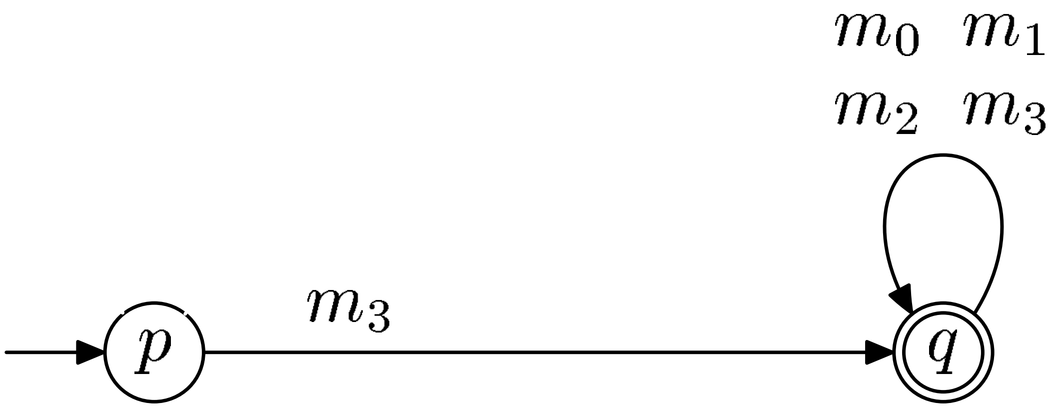
Let P be a PDS of size p, let $\mathcal{A}$ be a multi-automaton with q states.
The set of predecessors (successors) of $L(\mathcal{A})$ is also regular, and a multi-automaton recognizing it can be computed in $O(p \cdot q^2)$ time and $O(p \cdot q)$ space using a saturation algorithm

Let P be a PDS of size p, and let ϕ be an LTL-formula (simple valuations) whose negation corresponds to a Büchi automaton having b states.
Model-checking ϕ takes $O(p \cdot b^3)$ time and $O(p \cdot b^2)$ space.

References: [Büchi 64], [Bouajjani, E., Maler CAV97], [Finkel, Willems, Wolper INFINITY97], [E., Hansel, Rossmanith, Schwoon CAV00]
(See also [Alur, Etessami, Yannakakis CAV01], [Benedikt,Godefroid,Reps ICALP01])

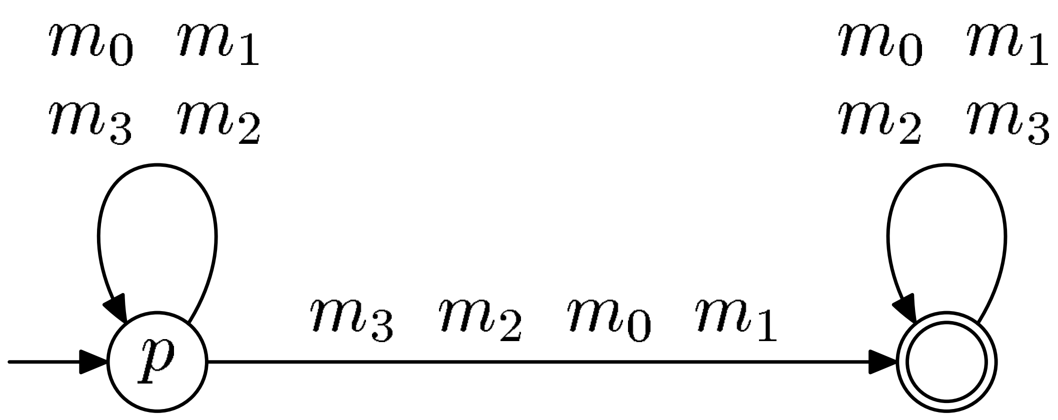
The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1 : \langle p, m_0 \rangle \xhookrightarrow{a} \langle p, m_1 \rangle$$
$$r_2 : \langle p, m_0 \rangle \xhookrightarrow{b} \langle p, m_2 \rangle$$
$$r_3 : \langle p, m_1 \rangle \xhookrightarrow{\epsilon} \langle p, m_0 m_3 \rangle$$
$$r_4 : \langle p, m_2 \rangle \xhookrightarrow{a} \langle p, m_3 \rangle$$
$$r_5 : \langle p, m_3 \rangle \xhookrightarrow{\epsilon} \langle p, \epsilon \rangle$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1: \langle p, m_0\rangle \xhookrightarrow{a} \langle p, m_1\rangle$$
$$r_2: \langle p, m_0\rangle \xhookrightarrow{b} \langle p, m_2\rangle$$
$$r_3: \langle p, m_1\rangle \xhookrightarrow{\epsilon} \langle p, m_0 m_3\rangle$$
$$r_4: \langle p, m_2\rangle \xhookrightarrow{a} \langle p, m_3\rangle$$
$$r_5: \langle p, m_3\rangle \xhookrightarrow{\epsilon} \langle p, \epsilon\rangle$$



Extensions

Counterexamples can be found by annotating new transitions with their ’causes’

Symbolic pushdown systems [E., Schwoon CAV01]

- Global variables encoded into control states $\rightarrow$ automata over very large alphabets
- Use symbolic automata with BDD-labelled transitions

Model-checking CTL* [E., Kucera, Schwoon TACS01]

- Reduced to LTL model-checking with regular valuations
- EXPTIME complete in the size of the PDS

Weighted PDSs [Reps, Schwoon, Jha SAS03]

– Each rule labelled with a ’weight’
– Modified saturation algorithm can solve generalized shortest path problems between sets of local configurations

Pushdown games [Cachat ICALP02]

– Saturation algorithm (for alternating pushdown systems) also basis of solution
– See also [Walukiewicz CAV97], [Alur, LaTorre,Madhusudan TACAS03]

Interprocedural dataflow-analysis

Goal: compute dataflow information for each node in the flow graph.

Direct mapping into model-checking problems. No coincidence theorems needed.

Bitvector problems (live variables, reaching definitions . . .), linear constant propagation.

If complete abstraction from data: 10^4 to 10^5 lines per minute.

References: [E., Knoop FOSSACS99], [Reps, Schwoon, Jha SAS03]

Model-checking (recursive) programs

Total correctness (including termination) for programs with finite datatypes but (possibly recursive) procedures.

Counterexamples are provided when property fails.

Finite automata for finitely encoding infinite sets of stack contents.

BDDs for concisely encoding finite sets of values of global and local variables.

Implemented in the Moped tool

References: [E., Schwoon CAV01], [Schwoon PhD 02]

Quicksort: Termination and correctness

N = number of bits for each array element
M = number of array elements

	$N = 3$ $M = 6$	$N = 4$ $M = 7$	$N = 1$ $M = 20$	$N = 1$ $M = 60$
globals	18	28	20	60
locals	18	24	26	31
time	12	396	6	562

Moped inside SLAM

SLAM toolkit developed by Ball and Rajamani at Microsoft Redmond

Creates Boolean Program as abstraction of C program using predicate abstraction

Model checks the abstraction using Bebop

Counterexample-guided abstraction refinement

Boolean programs have same expressive power as PDS

Converters implemented allowing to replace Bebop by Moped (option in SLAM).

Small C programs

Test suite of 64 small C programs (between 30 and 150 lines of code)

Various reachability properties checked in 178 iterations of SLAM

	Bebop		Moped	
time	209.7		46.4	
memory avg/max	4.4	9.0	4.3	6.8
BDD nodes avg/max (peak)	12.1	273	7.1	145
	average		maximum	
global variables	1.3		10	
max local variables	6.6		36	
max arguments	2.4		8	
max return values	1.7		6	

Large C programs: Drivers

Four drivers between 2200 and 7600 lines of code

Locking-unlocking properties checked in 32 iterations of SLAM

	Bebop		Moped	
time	944.2		88.5	
memory avg/max	10.7	13.1	7.6	12.8
BDD nodes avg/max (peak)	286.8	437	161.7	470
	average		maximum	
global variables	5.5		16	
max local variables	40.5		58	
max arguments	9.9		12	
max return values	12.0		15	

Very large C programs: A serial driver

27000 lines of code, errors found

	Bebop		Moped	
time	3488.1		50.7	
memory avg/max	24.4	25.9	23.2	25.5
BDD nodes avg/max (peak)	601.6	682	489.8	632
	average		maximum	
global variables	28.0		28	
max local variables	325.6		334	
max arguments	22.0		22	
max return values	27.0		27	

Security properties of C programs

MOPS: Tool for finding security bugs in C programs and verifying conformance to rules of defensive programming

Rules expressed as temporal logic (safety) properties $\rightarrow$ finite automata

- any call to `start(f)` should be followed immediately by a call to `open(f)`
- a `setuid-root` process should not be execute an untrusted program without first dropping its root privileges

References: [Chen, Wagner CCS02]

Security properties of Java programs

Security design of the JDK.

Method `CheckPermission`:

> Checkpoints at which stack contents are inspected.
> For a permission to be granted, all the methods in the call stack must have also been granted permission.

Security property $\rightarrow$ all stack contents belong to a certain regular language L.

Property violation: initial configuration is a predecessors of $\overline{L}$.

Requires to deal with atomic propositions interpreted over arbitrary regular sets of configurations.

References: [Jensen, LeMétayer, Thorn 99], [E., Kucera, Schwoon TACS01]

Interface compatibility

Interface: set of public methods of a module

Availability assumptions:

- An update method is only called after the corresponding initialization method

Modules A and B are compatible if there is an environment such that the guarantees of A meet the assumptions of B, and vice versa

Compatibility checking reduces to existence of winning strategy in a pushdown game with reachability winning condition

References: [Chakrabarti, de Alfaro, Henzinger, Jurdzinsky, Mang CAV02]

Authorisation problems

SPKI/SDSI trust management system

(Extended) Names: `K DE.ST.CS.faculty`

Certificates grant rights to names, or delegate granting them

Basic problem: is there a certificate granting right r to group g?

Advanced problems: is tehre a certificate that does not reveal sensitive information? Which is the shortest certificate?

Can be reduced to shortest path problem in weighted pushdown systems

References: [Reps, Schwoon, Jha 03]

Adding thread creation

Add rules

- $\langle p, X\rangle \hookrightarrow \langle p_1, w_1\rangle \parallel \langle p_2, w_2\rangle$
- $(\langle p_{1f}, \epsilon\rangle \parallel \langle p_{2f}, \epsilon\rangle) \hookrightarrow \langle p', w'\rangle$

Configuration $\Rightarrow$ tree over $\parallel$ and concatenation operators.

Finite coding of infinite sets of terms by finite tree automata.

(Representatives of) The set of predecessors (successors) of a regular set of (tree) terms can be computed in $O(n_\Delta \cdot n_L)$ time and space.

However: model-checking becomes undecidable

References: [Lugiez, Schnoebelen TCS 0?], [E., Podelski POPL00], [Bouajjani, Touili 03]

Adding thread synchronization

A communicating pushdown system is a tuple $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ of pushdown systems

A global configuration is a tuple $g = (c_1, \ldots, c_n)$ of configurations

$g_0 = (c_{10}, \ldots, c_{n0})$ is the initial global configuration

Let $g = (c_1, \ldots, c_n)$ and $g' = (c'_1, \ldots, c'_n)$

$$g \xrightarrow{\epsilon} g' \text{ if } \begin{array}{l} c_i \xrightarrow{\epsilon} c'_i \text{ for some } 1 \leq i \leq n \text{, and} \\ c'_j = c_j \text{ for every } j \neq i \end{array}$$

$$g \xrightarrow{a} g' \text{ if } \begin{array}{l} c_i \xrightarrow{a} c'_i \text{ and } c_j \xrightarrow{a} c'_j \text{ for some } i \neq j \text{, and} \\ c'_k = c_k \text{ for every } i \neq k \neq j \end{array}$$

Reachability in communicating pushdown systems

Undecidable, even for $n = 2$, $F = \{f\}$ [Ramalingam, TOPLAS 2000]

Our approach: Abstract path languages

Assume $F = F_1 \times F_2$

Idea: Compute abstract languages $A_1 \supseteq L(c_{01}, F_1)$ and $A_2 \supseteq L(c_{02}, F_2)$ such that $A_1 \cap A_2 \stackrel{?}{=} \emptyset$ is decidable

Then: If $A_1 \cap A_2 = \emptyset$, we have proved $L(c_{01}, F_1) \cap L(c_{02}, F_2) = \emptyset$

Formal framework: abstract interpretation

Algorithmic solutions for finite height abstractions and for commutative abstractions.

References: [Bouajjani, E., Touili POPL03]

Sponsor

The IST-FET ADVANCE project:

Advanced Verification Techniques for Telecommunication Protocols