Around Moped

Javier Esparza and Stefan Schwoon

Software Reliability and Security Group

University of Stuttgart

Survey of joint work with Tomáš Brázdil, Ahmed Bouajjani, Somesh Jha, Stefan Kiefer, Tony Kučera, Michael Luttenberger, Richard Mayr, Thomas Reps, Jan Strejček, Dejvuth Suwimonteerabuth

Initial motivation

Model checkers of the first generation (SPIN,SMV,Murphi, . . .) only work for flat finite-state systems.

Recursive procedural programs may be infinite-state,
even if all variables have a finite range (unbounded call stack).

Flattening of non-recursive procedural programs using inlining
may cause an exponential blow-up in the size of the program.

Our (initial) setup:

- Goal: Design model checkers that work directly on the procedural representation.
- Approach: Base as much as possible on automata theory.

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 $p\alpha$, where $p \in P$, $\alpha \in \Gamma^*$

Semantics: A (possibly infinite) transition system with configurations as states and transitions given by

$$\text{If} \quad pX \hookrightarrow q\alpha \in \delta \quad \text{then} \quad pX\beta \longrightarrow q\alpha\beta \quad \text{for every } \beta \in \Gamma^*$$

From programs to pushdown systems

State of a procedural program: $(g, (n, l), (n_1, l_1) \ldots (n_k, l_k))$, where

- g is a valuation of the global variables,
- n is the value of the program pointer,
- l is a valuation of local variables of the current active procedure,
- n_i is a return address, and
- l_i is a saved valuation of the local variables of the procedures on the call stack

Modelled as a configuration $pXY_1 \ldots Y_k$ where

$$p = g \qquad X = (n, l) \qquad Y_i = (n_i, l_i)$$

Correspondence between program statements and rules

procedure call	$pX \hookrightarrow qYX$
return	$pX \hookrightarrow q\varepsilon$
statement	$pX \hookrightarrow qY$

From the beginnings to MOPED

Formal model and first complexity results (1996 – 1997)

- Fundamental problem: manipulate infinite sets of configurations
- Key insight: use finite automata as data structure

Finding efficient algorithms (1999 – 2000)

- Efficient algorithms for computing *post** (and *pre**) of a regular set of configurations
- Polynomial algorithm in the size of the PDS for LTL model-checking

Dealing with the state explosion problem (2001 – 2002)

- Symbolic pushdown systems as modelling language
- Basic idea: use BDDs to compactly represent sets of PDS rules that differ only in their 'data part'

$$\langle g_1, \ldots g_k \rangle\ (n_1, \langle v_1, \ldots, v_l \rangle) \hookrightarrow \langle g'_1, \ldots, g'_l \rangle\ (n_2, \langle v'_1, \ldots, v'_k \rangle)\ (n_3, \langle v''_1, \ldots, v''_k \rangle)$$
$$(g_1 > v_2 \wedge v'_2 = v_1 \wedge v''_1 = v_2 \wedge \ldots)$$

MOPED (Stefan Schwoon): A model checker for pushdown systems (2002)

MOPED's input language

```
define DEFAULT_INT_BITS N    //N and M are parameters
int a[1,M];

module void qs (int left, int right) {
  int lo, hi, piv;

  if ::  (left >= right) -> return;
     ::  else            -> lo = left, hi = right, piv = a[right];
  fi

  do ::  (lo > hi)                        -> break;
     ::  ((lo <= hi) && (a[hi] > piv))  -> hi = hi - 1;
     ::  ((lo <= hi) && (a[hi] <= piv)) ->
                a[lo] = a[hi]; a[hi] = a[lo]; lo = lo + 1;
  od

  qs(left, hi);  qs(lo, right);
}

module void main () {
  qs(1,M);
  if ::  (E i (1,M-1) a[i] > a[i+1]) -> error:  goto error;
     ::  else -> ok:  goto ok;
  fi
}
```

Extending MOPED (2003-2005)

Theory and algorithms:

- Weighted Pushdown Systems
- Concurrent (Dynamic) Pushdown Systems
- Probabilistic Pushdown Systems
- CEGAR for Symbolic Pushdown Systems

Applications:

- Model-checking/symbolic testing of Java
- Program analysis
- Authorization problems

Weighted Pushdown Systems

Attach weights to the rules of a pushdown system

Weight of a path: sum of the weights of the rules used

Weight of a bundle of paths: minimum of the path weights

Basic result (SCP ’05): extension of the *post**-algorithm to obtain for each reachable state the length of a shortest path leading to the state.

Generalization:	$+$ / min	$\longrightarrow$	arbitrary semiring
	shortest path	$\longrightarrow$	‘summary’ of all paths

Implemented for abstract semirings

Concurrent (Dynamic) Pushdown Systems

Goal: formal models for procedural multithreaded programs

Synchronous communication (POPL'03)

- Model: pushdown systems communicating through rendezvous
- Problem: deciding properties of the intersection of context-free languages
- Main result: commutative abstractions are decidable

Asynchronous communication (FSTTCS'05)

- Follow-up to work by Qadeer and Rehof in TACAS '05
- Model: pushdown systems communicating through shared variables
- Approach: compute underapproximations of the reachable states (split runs into 'contexts' during which only one thread writes to shared variables)
- Result: extension of the basic algorithm to compute the reachable states for up to *n* contexts

CEGAR for symbolic pushdown systems

Still unpublished

Only variables with a finite range

Modification of the basic *post**-algorithm to find dags of counterexamples

BDD-based Craig-interpolation:

- Weakest and strongest interpolants naturally computed using quantifier elimination
- Can be computed while determining if counterexample is spurious (connection to Hoare proofs)
- New ‘good’ interpolants: conciliated interpolants

Currently working on: Finding suitable applications!

NDD-based Craig-interpolation

Probabilistic Pushdown Systems

Attach probabilities to pushdown rules

Interesting class of infinite-state Markov chains

Model-checking both for linear and branching-time (LICS'04)

- Will the program terminate with probability 1?
- Is the probability that a request never gets granted below 0.01?

Expectations and variances of service times (LICS'05, FOCS'05)

- What is the probability that the average service time of a run is between 30 and 32 seconds?

Look also for work by Etessami and Yannakakis

Current work: Approximation algorithms

Model-checking/symbolic testing of Java

Goal: create a Java front-end for Moped

Translation starts at bytecode level; captures a large subset of Java's capabilities

Restrictions:

- variables limited to finite range
- finite heap size
- some advanced language features

$\longrightarrow$ symbolic testing for a large set of inputs simultaneously

Implementation: JMoped (TACAS'05)

Currently working on:

- Combine with CEGAR approach
- Testing environment with a GUI

Program analysis

Idea: Weighted PDS yield new unified framework for data-flow analysis problems (SAS'03, SCP'05)

Abstract semirings can encode many different data-flow problems
(from bitvector problems to affine-relation analysis).

pre*/*post** primitives compute data-flow values for each configuration

Previous methods "merge" values at a program point regardless of calling context; WPDS allow to make queries w.r.t. specific stack configurations.

Can compute an example path (or *set* of paths) that "explains" the computed data-flow values (contribution to program understanding).

Implementation: WPDS library (used in *Codesurfer*)

Authorization problems

PDS can be used to model the SPKI/SDSI authorization framework (Jha/Reps 2002).

SPKI/SDSI:

Principals $\widehat{=}$ Public Keys $\widehat{=}$ PDS control locations

"Extended names" provide a hierarchical name space, e.g.

$$K_{Alice}\ \texttt{friends} \qquad K_{university}\ \texttt{institutes staff}$$

Name space hierarchy $\widehat{=}$ stack symbols

Authorization certificates $\widehat{=}$ pushdown rules, e.g.

$$K_{library}\ \square \hookrightarrow K_{university}\ \texttt{institutes staff}\ \blacksquare$$

pre*/*post** primitives compute the set of authorized principals

Weights used to express additional properties in certificates: (CSFW'03)

Detailed access rights (read/write etc; powerset domain)
"Find me a set of certificates giving me as many rights as possible."

Privacy/sensitivity information
"Prove my access rights while trying not to reveal unnecessary information."

Recency/Validity
"Prove my access rights with recent certificates/certs that are valid as long as possible."

Distributed certificate chain discovery (search distributed among several certificate servers)

Current work: Embedding into Kerberos services

Generalization to multiple authorization