Using Automated Theorem Proving within the GECCOO Project

Sorin STRATULAT

May, 2004

Motivation

Theorem proving for real applications, e.g. JavaCard bytecode certification

Certicartes

Specification of Defensive VM (DVM) and Bytecode Verifier (BV)

- Specification of Offensive VM (OVM) and Abstract VM (AVM)
- Certification of i) OVM and ii) BV (AVM, monotonicity)

☞ by using the COQ Theorem Prover. Disadvantages:

- highly interactive
- script-based
- expensive error recovery

☞ by Automated Theorem Proving

Goal: Use this experience in the GECCOO project

Overview

I. The SPIKE Theorem Prover

- Specifications
- Inference System

II. Specification and Validation of JavaCard VMs

- Extensions of SPIKE
- Experimental Results

III. Working Plans

SPIKE: Settings

$$\left.\begin{array}{l}\text{axiomes } Ax\text{, conjecture } \phi \\ \text{relation de conséquence } \models\end{array}\right\} \; Ax \models^{?} \phi \quad \Leftrightarrow \quad Ax \models \phi\gamma$$

Problem: potential infinite number of closed instances $\phi\gamma$

☞ induction + reasoning techniques

Descente infinie" Principle

[Fermat (1659)]

$\mathcal{P}$ (infinite) set of formulas, $<$ Noetherian order over $\mathcal{P}$

1. for any counterexample from $\mathcal{P}$ there exists a smaller one

---

2. $\mathcal{P}$ has only true formulas

Applications :

- implicit induction provers
- saturation-based provers

Methodology for building provers

[Stratulat, 2000]

Sound and modular "descente infinie"-based theorem provers, providing:

✔ clear separation between logic and computation

✔ easy and sound extensions

✔ inference systems "à la carte"

SPIKE

✔ Specification

- first-order logic
- constructor-based many-sorted conditional specifications
- initial consequence relation
- syntactic order over the conditional equations

✔ Proofs

- implicit induction + deduction techniques (rewriting, subsumption, tautology elimination,...)

SPIKE: Specifications

```
specification : add
% problem description
sorts : nat ;
constructors :
0    :          → nat;
s_   : nat      → nat;
defined functions:
_+_ : nat nat → nat;
axioms:
0 + x = x;
s(x) + y = s(x+y);
% proof part
strategy: ...
conjectures: ...
```

SPIKE: The Inference System

- instance of an abstract inference system [Stratulat, 2000; Stratulat, 2001]
- deduction techniques

tautology elimination

elimination of subsumed clauses

C' subsumes C if $C'\sigma$ is a subclause of C

unconditional rewriting

$C[t]_p \rightarrow_{Ax\langle \mathsf{H} \rangle} C[t']_p$ iff $t \rightarrow_{Ax\langle \mathsf{H} \rangle} t'$ and $C[t']_p \prec C[t]_p$

case analysis

rules $\cup_{i=1}^{n} \{P_i \Rightarrow l_i \rightarrow r_i\}$ and $C[l_i\sigma_i]_{p_i}$

$\mathsf{CaseAnalysis}(C) \triangleq \cup_i \{P_i\sigma_i \Rightarrow C[r_i\sigma_i]_{p_i}\}$ if $Ax \models_{ini} \bigvee_i P_i\sigma_i$

SPIKE: Proofs

Ax axioms, E conjectures, H premises

inference rule $(E,\ H) \vdash (E',\ H')$

SPIKE: Proofs

Ax axioms, E conjectures, H premises

inference rule $\quad (E,\ H) \vdash (E',\ H')$

derivation $\quad (E^0,\ H^0) \vdash \ldots \vdash (E^n,\ H^n) \ldots$

SPIKE: Proofs

Ax axioms, E conjectures, H premises

inference rule $(E,\ H) \vdash (E',\ H')$

derivation $(E^0,\ H^0) \vdash \ldots \vdash (E^n,\ H^n) \ldots \hookrightarrow^{E^n=\emptyset}$ success

SPIKE: Proofs

Ax axioms, E conjectures, H premises

inference rule $(E,\ H) \vdash (E',\ H')$

derivation $(E^0,\ H^0) \vdash \ldots \vdash (E^n,\ H^n) \ldots \hookrightarrow^{E^n=\emptyset}$ success $\hookrightarrow^{H^0=\emptyset}$ proof

SPIKE: Proofs

Ax axioms, E conjectures, H premises

inference rule $(E,\ H) \vdash (E',\ H')$

derivation $(E^0,\ H^0) \vdash \ldots \vdash (E^n,\ H^n) \ldots \hookrightarrow^{E^n=\emptyset}$ success $\hookrightarrow^{H^0=\emptyset}$ proof

Properties

soundness

$(E,\ \emptyset) \vdash \ldots \vdash (\emptyset,\ H) \quad \Rightarrow \quad Ax \models E$

refutational soundness

$(E,\ \emptyset) \vdash \ldots \vdash (E',\ H) \qquad Ax \not\models E' \quad \Rightarrow \quad Ax \not\models E$

SPIKE: Proofs

Ax axioms, E conjectures, H premises

inference rule $(E,\ H) \vdash (E',\ H')$

derivation $(E^0,\ H^0) \vdash \ldots \vdash (E^n,\ H^n) \ldots \hookrightarrow^{E^n=\emptyset}$ success $\hookrightarrow^{H^0=\emptyset}$ proof

Properties

soundness

$$(E,\ \emptyset) \vdash \ldots \vdash (\emptyset,\ H) \quad \Rightarrow \quad Ax \models E$$

refutational soundness

$$(E,\ \emptyset) \vdash \ldots \vdash (E',\ H) \qquad Ax \not\models E' \quad \Rightarrow \quad Ax \not\models E$$

A derivation can

- terminate with success,
- terminate with failure (excerpt of a counterexample)
- diverge

SPIKE Inference System (official version)

PROTHEO group, LORIA [Bouhoula, 1997]

Generate $(E \cup \{C\},\ H) \vdash (E \cup (\cup_\sigma E_\sigma),\ H \cup \{C\})$
if for any test substitution σ of C

- either $C\sigma$ is a tautology and $E_\sigma = \emptyset$,
- or $C\sigma \rightarrow_{Ax\langle H \cup E \cup \{C\}\rangle} C'$ and $E_\sigma = \{C'\}$
- otherwise, $E_\sigma =$CaseAnalysis$(C\sigma)$

Case Simplify $(E \cup \{C\},\ H) \vdash (E \cup E',\ H)$
if $E' =$CaseAnalysis(C)

Simplify $(E \cup \{C\},\ H) \vdash (E \cup \{C'\},\ H)$
if $C \rightarrow_{Ax\langle H \cup E\rangle} C'$

Subsume $(E \cup \{C\},\ H) \vdash (E,\ H)$
if C is subsumed by another clause from $Ax \cup H \cup E$

JavaCard Specifications

Getting SPIKE specifications from COQ

The converter is written in OCAML using the $camlp4$ preprocessor.

COQ Specification

```
Definition CONV := [t,t':type_prim][state:jcvm_state]
Cases (stack_f state) of
(cons h lf) ⇒
  Cases (extr_from_opstack t (opstack h)) of
  (Some (k, lv)) ⇒ (update_frame (push_opstack (
           vPrim (tpz2vp t' (t_convert t t' k))) lv h) state) |
  None ⇒ (AbortCode opstack_error state)
  end |
_ ⇒ (AbortCode state_error state)
end.
```

OCAML Specification

☞ COQ extraction commands

```
let cONV t t' state =
  match stack_f state with
    Nil → abortCode State_error state
  | Cons (h, lf) →
      (match extr_from_opstack t (opstack h) with
         Some p →
           (let Pair (k, lv) = p in
           update_frame
             (push_opstack (VPrim (tpz2vp t'
                (t_convert t t' k))) lv h) state)
       | None → abortCode Opstack_error state)
```

SPIKE Specification

☞ the converter OCAML $->$ SPIKE

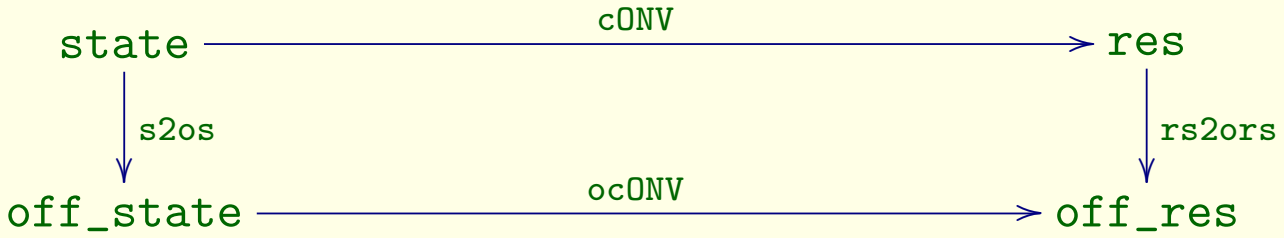
```
axioms : stack_f( state) = Nil
         ⇒ cONV( t, t', state) = abortCode( State_error, state);

         stack_f( state) = Cons( h, lf),
         extr_from_opstack( t, opstack( h)) = Some( Pair( k, lv))
         ⇒ cONV( t, t', state) = update_frame( push_opstack(
           VPrim( tpz2vp( t', t_convert( t, t', k))), lv, h), state);

         stack_f( state) = Cons( h, lf),
         extr_from_opstack( t, opstack( h)) = None
         ⇒ cONV( t, t', state) = abortCode( Opstack_error, state);
```

DVM - OVM Commutation Proofs

Input: offensive and defensive machines

Output: each instruction commutes (45 instructions)

```
conjectures :
        state = Build_jcvm_state( sh, hp, Cons (h, lf)),
        res = cONV(n, z0, state) ⇒
        res = abortCode( Type_error, state),
        res = abortCode( Signature_error, state),
        rs2ors( res) = ocONV (n, z0, s2os(state));
```



Proofs by:
- case analysis
- induction

Features

- sufficient complete and ground convergent
- parameterized specification (lists, pairs)
- existential variables (OCAML local variables)
- "big" specification

  61 sorts, 164 constructors, 446 defined functions, $\sim$ 2200 rules (February 2002)

SPIKE Improvements: Existential Variables

```
stack_f( state) = Cons( h, lf),
extr_from_opstack( t, opstack( h)) = Some( Pair( k, lv))
 ⇒ cONV( t, t', state) = update_frame( push_opstack(
    VPrim( tpz2vp( t', t_convert( t, t', k))), lv, h), state);
```

1. conditional equations cannot be oriented into rewrite rules

   a) use premises only in "safe" proof fragments

2. Generate failure

$$eq(O,O) = True \qquad eq(O,S(x)) = False$$
$$eq(S(x),O) = False \quad eq(S(x),S(y)) = eq(x,y)$$

   ☞ $eq(a,O)$ cannot be instanciated

   a) generalization of existential variables in universal variables

   b) definition changement, then case analysis:

$$x = O \wedge y = O \Rightarrow eq(x,y) = True \qquad x = O \wedge y = S(z) \Rightarrow eq(x,y) = False$$
$$x = S(z) \wedge y = O \Rightarrow eq(x,y) = False \quad x = S(z) \wedge y = S(t) \Rightarrow eq(x,y) = eq(z,t)$$

SPIKE Improvements: Dealing with "Big" Specifications

Need of automatic and efficient operations:

- extraction of subspecifications for each instruction
- recording of previous failed operations
- "fast" versions of Generate and Case Simplify
- specific simplification techniques

New Generate Rule

☞ test substitutions by unification

Generate $(E \cup \{C\},\ H) \vdash (E \cup (\cup_\sigma E_\sigma),\ H \cup \{C\})$

if $\exists t \in C$

- having induction variables and
- $\exists a \in Ax$ defining $t\#head$ and $(\sigma, \sigma_a) = mgu(t, a)$ and
- $C\sigma[t] \rightarrow_{\{a\sigma_a\}} C'$ and $E_\sigma = \{C'\}$

```
%  **** l_update_nth ****
l_update_nth( Nil, n, a) = Nil;
l_update_nth( Cons( x0, xs), 0, a) = Cons( a, xs);
l_update_nth( Cons( x0, xs), S( n'), a) = Cons( x0, l_update_nth( xs, n', a))
```

$l\_update\_nth(u_1, u_2, u_3)$ has the test substitutions

$$\{\langle u_1 \leftarrow Nil\rangle, \langle u_1 \leftarrow Cons(u_4, u_5), u_2 \leftarrow O\rangle, \langle u_1 \leftarrow Cons(u_4, u_5), u_2 \leftarrow S(u_6)\rangle\}$$

☞ faster (σ and $a\sigma_a$ is directly computed from mgu)

☞ less new conjectures ($\sim$ 10 w.r.t. $\sim$ 1000 for the old version)

New Case Simplify Rule

☞ no $\bigvee_i P_i\sigma_i$ test for terms with no induction variables.

Example:

```
P(0) = True     ⇒ f(0) = True;
P(S(x)) = True  ⇒ f(S(x)) = True;
P(x) = False    ⇒ f(x) = False;
```

☞ no application on $f(x)$

Specific Simplification Techniques

- congruence closure on conditions

  $(E \cup \{a = b \wedge \ldots \wedge b = c \wedge \ldots \Rightarrow \ldots\}, H) \quad \vdash$

  $(E \cup \{a = b \wedge \ldots \wedge b = c \wedge \ldots \wedge a = c \Rightarrow \ldots\}, H)$

- auto simplification with rules $e \to t$

  $(E \cup \{e = t \wedge \ldots \wedge s[e] \wedge \ldots \Rightarrow \ldots\}, H) \quad \vdash$

  $(E \cup \{e = t \wedge \ldots \wedge s[t] \wedge \ldots \Rightarrow \ldots\}, H)$

- augmentation with lemmas $C_1 \Rightarrow C_2$

  $(E \cup \{C_1' \Rightarrow C_2'\}, H) \quad \vdash (E \cup \{C_1' \wedge C_2\sigma \Rightarrow C_2'\}, H)$

  if C_1 is subsumed by $C_1' \Rightarrow C_2'$ with substitution σ

Experiments

Proofs [Barthe, Stratulat 2003]

- DVM - OVM: 41 (21 zero-knowledge proofs)
- DVM - AVM: 17
- Monotonicity: 21

Interaction of the user with SPIKE by

- fixing a precedence over the function symbols (for Generate and rewriting)
- providing intermediate lemmas (needs deep knowledge of the problem)
- changing the proof strategy (very few cases)
- changing function definitions (very few cases)

Conclusions wrt Certicartes project

- industrial applications with "home-made" automated theorem provers
  - access to internal structure
  - shaping according to needs (by using the building methodology)
- successful use of SPIKE
  - first full first-order specification for JavaCard platform
  - VMs certification
  - detection of errors and excerpt of counterexamples
  - high degree of automatization
  - efficiency ($<$ 4 hours for the DVM-OVM proofss on a machine with two 1GHz Intel Pentium III processors $+$ 896 Mbytes RAM)
- dramatic time reduction in the design of certified specifications

Working Plans

☞ following a similar working methodology for the GECCOO case studies.

SPIKE improvements

- more user interaction (treatment of difficult cases)
- more convivial user interface (graphical interface, shorter proof traces)

Integration of SPIKE with other GECCOO tools

- haRVey ?
- SPIKE plugin for ECLIPSE

Automated Theorem Proving for JavaCard Bytecode Certification

Sorin STRATULAT

References

[Bouhoula, 1997] A. Bouhoula. Automated theorem proving by test set induction. Journal of Symbolic Computation, 23:47–77, 1997.

[Stratulat, 2000] S. Stratulat. Preuves par récurrence avec ensembles couvrants contextuels. Applications à la vérification de logiciels de télécommunications. PhD thesis, Université Henri Poincaré, Nancy I, 2000.

[Stratulat, 2001] S. Stratulat. A general framework to build contextual cover set induction provers. Journal of Symbolic Computation, 32(4):403–445, 2001.

Content of the talk