(Deeply) Integrating Proving and Programming

Joseph Kiniry
KindSoftware Research Group,
a part of the Systems Research Group
University College Dublin
Dublin, Ireland
Funding under EU FP6 FET project "Mobius"

joint work with Cees-Bart Breunesse, Julien Charles,
Radu Grigore, and Sam Owre

Provers and Programmers

- these two communities rarely interact
- most programmers do not prove, and most provers do not program
- but formal specifications are of comparable size and complexity to program (components) these days
- why not exchange the best ideas from each community and build a platform for such?

Overall Goal

- always use the appropriate tool for the job
- tie together appropriate “small” systems
- build platform for specification and composition of formal-centric components
- i.e., this is an experimental platform realizing the vision of the the verified software community & Rusby’s “evidential toolbus”

Original Context of Work

- Security of Systems (SoS) group (née LOOP, Logic of Object-Oriented Programs) in Nijmegen's Loop compiler
  - (full, sound, coalgebraic) semantics of Java and JML in ~50K of PVS
  - generate 10Ks of specifications and a handful of lemmas for a given verification
  - proof states grew to multiple GB in size
- ESC/Java2

Current Context

- Mobius EU FP6 FET project
  - 16 partners focusing on logic- and type-based verification of (concurrent and single-threaded) JML-annotated Java code at the source and bytecode level using proof-carrying/evidence-carrying code
  - UCD in charge of all tool development
- Hoare's Verified Software Grand Challenge
  - contribute code, tools, proofs, etc.

Our Work on PVS and Motivations

- while at Radboud University Nijmegen (Bart Jacob's LOOP/SoS group)
- initial work "porting" PVS to an Open Source Common Lisp system
- aforementioned extensions to foundation and interface
- the Loop tool: verification system for JML-annotated Java

PVS, SAL, and Yices

- what does this have to do with SRI's tools?
- we use and extend PVS heavily
- we next will incorporate SAL and Yices in to ESC/Java2 and/or the Mobius PVE
- we find PVS is an excellent platform for experimentation because natural syntax for programmers and its "frankenstein" nature

Initial Experimentation

- Emacs-centric experimentation
- take best components for software development in Emacs & apply them to PVS
- new features: theory browsing, non-ASCII fonts, customized theory and sequent pretty-printing, context awareness, type-aware completion, smart cut-and-paste, implicit hyperlinking, etc.

emacs@trystero.kindsoftware.com

PVS File Edit Options Buffers Tools X-Symbol Help

```
% booleans declares the type 𝔹 and its abbreviation 𝔹, along
% with the 𝔹 constants true and false and the 𝔹 connectives.
% The properties of the connectives are given later, but the connectives
% are built in to the typechecker so must be provided early on.
% Note: the 𝔹 type could be defined as the enumeration type {false,
% true}, but booleans are primitive; the correct handling of enumeration
% types requires the 𝔹 type.

booleans: THEORY
 BEGIN

  𝔹: NONEMPTY_TYPE
  𝔹: NONEMPTY_TYPE = 𝔹
  ⊥, ⊤: 𝔹
  ¬: [𝔹 ⟶ 𝔹]
  ⊗, &, ⊕, ⟹, ⇒, ⟵, ⇔, ⟺: [𝔹, 𝔹 ⟶ 𝔹]

 END booleans

% equalities contains the declaration for =.  It has a single type
% parameter.  Properties of equality are given in equality_props.

equalities [T: TYPE]: THEORY
 BEGIN

  =: [T, T ⟶ 𝔹]

 END equalities


notequal[T: TYPE]: THEORY
 BEGIN
  x, y: VAR T

  ≠(x, y): 𝔹 = ¬ (x = y)
 END notequal
```

-0:%* **prelude.pvs** 1% L34 (PVS :ready XS:pvs/s)----------

Systems Research Group
School of Computer Science and Informatics
University College Dublin

emacs@trystero.kindsoftware.com

PVS File Edit Options Buffers Tools Xref ECB Help

```
bitvectors
finite_sets
loop_prelude
character_adt
list_adt
ordstruct_adt
prelude.pvs
 Theories
  booleans
   Types
    bool
    boolean
   Constants
    FALSE
    TRUE
   Formulas
    NOT [boolean -> boolean]
    AND [boolean, boolean -> boolean]
    & [boolean, boolean -> boolean]
    OR [boolean, boolean -> boolean]
    IMPLIES [boolean, boolean -> boolean
    => [boolean, boolean -> boolean]
    WHEN [boolean, boolean -> boolean]
    IFF [boolean, boolean -> boolean]
    <=> [boolean, boolean -> boolean]
  equalities[T: TYPE]
  notequal[T: TYPE]
  if_def[T: TYPE]
  boolean_props
   Postulates
    bool_exclusive
W-0 prelude.pvs
```

```
character_adt.pvs
list_adt.pvs
ordstruct_adt.pvs
prelude.prf
prelude.pvs
pvs-language.help
pvs-prover.help
pvs.bnf
pvs.grammar
pvs.help
strategies.lisp
W-1 ...oS/PVS/lib
```

```
prelude.pvs|lib
character_adt.pvs
mod.pvs
ordstruct_adt.pvs
list_adt.pvs
prelude.pvs
speedbar_elements.txt
pvs.wy
pvs-grammar.el
func_composition.pvs
top.pvs|finite_sets
sums.pvs
W-2 History
```

```
% The PVS prelude.

% The prelude consists of theories that are built in to the PVS system.
% It is typechecked the same as any other PVS theory, but there are hooks
% in the typechecker that require most of these theories to be available,
% hence the order of the theories is important.  For example, no formulas
% may be declared before the booleans are available, as the formula is
% expected to have type bool.  Since definitions implicitly involve both
% formulas and equality, the booleans theory may not include any
% definitions.  Formulas are given below as AXIOMs, POSTULATEs, and LEMMAs.
% POSTULATEs are formulas that can be proved using the decision procedures,
% but would have to be given as axioms in a pure development of the theory.
% AXIOMs are formulas that cannot be proved, and LEMMAS are formulas that
% have been proved.

% booleans declares the type boolean and its abbreviation bool, along
% with the boolean constants true and false and the boolean connectives.
% The properties of the connectives are given later, but the connectives
% are built in to the typechecker so must be provided early on.
% Note: the boolean type could be defined as the enumeration type {false,
% true}, but booleans are primitive; the correct handling of enumeration
% types requires the boolean type.

booleans: THEORY
 BEGIN

  boolean: NONEMPTY_TYPE
  bool: NONEMPTY_TYPE = boolean
  FALSE, TRUE: bool
  NOT: [bool -> bool]
  AND, &, OR, IMPLIES, =>, WHEN, IFF, <=>: [bool, bool -> bool]

 END booleans


% equalities contains the declaration for =.  It has a single type
% parameter.  Properties of equality are given in equality_props.

equalities [T: TYPE]: THEORY
 BEGIN

  =: [T, T -> boolean]

 END equalities


notequal[T: TYPE]: THEORY
 BEGIN
  x, y: VAR T
```

--:-- **prelude.pvs|lib** Top (14,19) (PVS :exit)----Fri Apr 23 11:17AM 0.67------

Problems with Emacs-based Version

- not “componentized”
- not a good platform for integration experimentation
- fragile and slow due to implementation in elisp (used wisent package = bison in elisp)
- Emacs as an interface/foundation inspires both devotion and revulsion

A Small Automatic Prover Interface

- formally specified interface for arbitrary provers performing fully automatic actions
- small interface to configure and control automated prover
- provided with demonstration implementations in Java, C, and OCaml

Tackle the Platform Problems First

- develop a generic component platform for formal tools
- derive platform from concrete experiences
- first case study is in generic interfaces to automated provers for ESC/Java2
- deal with interface and “sales” issues later

Automated Provers Interface

- only eight methods
  - start_prover and stop_prover
  - set_prover_resource_flags
  - signature
  - declare_axiom
  - make_assumption and retract_assumption
  - is_valid

Interface Design Constraints

- resource specifications
  - general-purpose time, space, memory, and prover behavior hinting
- signature and theory independence
  - formally contextualize prover use
- assumption manipulation
  - retractions essential for primary use cases

Realizations of Interface

- implementations now available/ongoing for Simplify, CVC3, PVS, and Coq
- experimentation with multiple object logics and derivation of logics for different provers from “canonical” PVS theory
- all being implemented as Eclipse plugins

Interactive Provers Interface

- challenge: provide the same (lightweight) interface for an interactive theorem prover
- observation: user interactions are not (significantly) different than programmatic interactions
- solution: model the software interface according to using user behavior

Prover/Editor

- lightweight interactive prover framework
- implemented as an Eclipse plugin
- provides framework for common “perspectives” (i.e., theory browsing, syntax highlighting, preferences, contextual documentation, etc.) and prover interaction
- experimentation has shown that incorporating a new prover requires only couple hundred lines of code

File Edit Navigate Search Project Run Window Help

Package Explorer

- bicolano [svn+ssh://jcharles@scm.g
- Coq [trunk]
  - exemples
  - exemples-mariela
  - JmlSpecs
  - kildall
  - PileIP
- QuickSort
- sandbox
- vcGen-wpMod [coq-proofs/vcGen-wp
- verifier_with_models

Domain.v

```
(* <Insert License Here>

   $Id: Domain.v 69 2006-03-06 20:16:11Z davidpichardie $ *)

(** Formalization of Java semantic domain.
 Based on The "Java (TM) Virtual Machine Specification, Second Edition,
  Tim Lindholm, Frank Yellin"

 @author David Pichardie, ...  *)

Require Import Program.
Require Export ZArith.
Require Export List.
Open Scope Z_scope.

(** All semantic domains and basic operation are encapsulated in a module si

Module Type SEMANTIC_DOMAIN.

 (** We depend on the choices done for program data structures *)
 Declare Module Prog : PROGRAM. Import Prog.

 (** Common interface for numerics domains *)
 Module Type NUMERIC.
  Parameter t : Set.
  Parameter toZ : t -> Z.
  Parameter fromZ : Z -> t.
 End NUMERIC.

 Declare Module Byte  : NUMERIC.
 Declare Module Short : NUMERIC.
 Declare Module Int   : NUMERIC.

 (** range of numerics interpretations *)
 Parameter byte_range : forall b, -2^7 <= Byte.toZ b < 2^7.
 Parameter short_range : forall s, -2^15 <= Short.toZ s < 2^15.
 Parameter int_range : forall i, -2^31 <= Int.toZ i < 2^31.
 (* Remark: we don't need surjection, it will be a consequence of furthers d

 (** conversion *)
```

Outline

- Domain.v
  - Module Type SEMANTIC_DOMAIN.
    - Declare Module Prog : PROGRAM.
    - Module Type NUMERIC.
      - Parameter t
      - Parameter toZ
      - Parameter fromZ
    - Declare Module Byte : NUMERIC.
    - Declare Module Short : NUMERIC.
    - Declare Module Int : NUMERIC.
    - Parameter byte_range
    - Parameter short_range
    - Parameter int_range
    - Parameter b2i
    - Parameter b2i_prop
    - Parameter s2i
    - Axiom s2i_prop : forall s, Short.
    - Parameter i2b
    - Parameter i2b_prop1
    - Parameter i2b_prop2
    - Parameter i2s
    - Parameter i2s_prop1
    - Parameter i2s_prop2
    - Parameter i2bool
    - Parameter iadd
    - Parameter iadd_prop1
    - Parameter iadd_prop2
    - Parameter iadd_prop3
    - Parameter isub
    - Parameter imul
    - Parameter idiv

Problems | Javadoc | Declaration | Prover Editor Toplevel

```
stack2locvar_prop1 is assumed

Editing file:
Domain.v
Interactive Module Type SEMANTIC_DOMAIN started
Module Prog is declared
```

Eclipse-independence

- Eclipse, Eclipse, Eclipse... enough about Eclipse!
- I agree!
- each subsystem is implemented as
  - a formally specified component
  - a command-line tool (e.g., for composition with pipes)
  - standard GNU interface (e.g., for Emacs)

Other Components Under Development

- automated provers: Simplify, SMT-LIB solvers like CVC3, Yices
- PVS, Coq, and Isabelle
- BML and Umbra
- VC generators and wp/sp calculi
- JML tool suite and ESC/Java2
- proof-transforming compiler

Challenge to the Community

- beg, borrow, or steal the best ideas from the software communities for your tools
- focus more on usable and useful interfaces (both GUI and API)
- eat your own dogfood
- get involved in the various “evidential toolbus” efforts (i.e., SRI’s stuff and the Mobius platform)