AFM, Seattle 21 August 2006

Automated Formal Methods 2006
Welcome and Introduction

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Welcome to AFM 2006

Agenda

- This introduction
- Overviews/news about PVS, SAL, Yices
- Lunch
- Invited talk: Joseph Kiniry
- Contributed talks and papers

Introduction

- Goal: dialog between developers and users
- Two-slide introductions to PVS, SAL, Yices
- Responding to changes in the world around us

PVS in Two Slides

- Comprehensive system for formal specification and analysis
- Specification language based on higher order logic extended with dependent types and structural and predicate subtypes, and includes constructs for recursively defined abstract data types, recursive functions, inductive relations, and tabular specifications, as well as traditional logical formulas
- Analysis capabilities include very strong typechecking (which can involve theorem proving), direct execution (at speeds within a factor of five of hand-crafted C), theorem proving, and symbolic model checking (with predicate abstraction)
- The PVS theorem prover provides powerful automation including rewriting and decision procedures for real and integer arithmetic, and is scriptable (strategies)

PVS in Two Slides (ctd.)

- Properties to be verified can be expressed as individual logical formulas, as CTL properties (for model checking), or as theory interpretations
- The system is supported by massive built-in and user-provided libraries of specifications for mathematics and computer science
- New in PVS 4.0: random testing, Yices as endgame prover, language extensions, port to CMU Lisp and new platforms

SAL in Two Slides

- PVS can be applied to any formalized mathematics
- SAL is specialized to the specification and analysis of state machines and can therefore apply more targeted automation
- SAL is supported by a suite of tools based on state-of-the-art model-checking technology for LTL properties (and CTL on the common fragment) but, unlike other model checkers, it is not restricted to finite-state state systems
- Unlike most other languages for model checkers, SAL supports fairly high-level specifications, with a type system similar to that of PVS; language uses guarded commands (good for protocols, not for sequential programs), supports synchronous and asynchronous composition of modules
- SAL is scriptable (in Scheme)

SAL in Two Slides (ctd.)

- For finite state systems, SAL provides both a symbolic model checker (using BDDs) and a bounded model checker (using a SAT solver), and a prototype “witness” model checker; there’s also a deadlock checker and simulator
- For infinite state systems SAL provides an “infinite bounded” model checker that uses SMT solving
- The bounded model checkers can verify safety properties by k-induction and can use lemmas
- The infinite bounded model checker blurs the line between theorem proving and model checking
- New in SAL 2.4: Yices as default SAT and SMT solver (several others also supported); concrete counterexamples from infinite bounded model checker when using Yices; application to test generation

Yices in Two Slides

- One of the strengths of PVS has been its decision procedures for theories such as linear arithmetic and equality with uninterpreted functions
- Yices makes these capabilities available to other applications in an extended and vastly more powerful form
- Whereas decision procedures consider only conjunctions of terms, Yices integrates decision procedures with SAT solving and is able to handle arbitrary propositional combinations of terms in its decided theories
- It's an SMT solver (satisfiability modulo theories)
  - Won every division in the CAV SMT competition

Yices in Two Slides (ctd.)

- Yices decides formulas in the combined theories of linear arithmetic over integers and reals (including mixed forms), fixed size bitvectors, equality with uninterpreted functions, recursive datatypes (such as lists and trees), extensional arrays, dependently typed tuples and records of all these, lambda expressions, and some quantified formulas
- Decides whether formulas are unsatisfiable or satisfiable, and in the latter case it can construct an explicit satisfying instance
- For unsatisfiable formulas, it can optionally find an assignment that maximizes the weight of satisfied clauses (i.e., MaxSMT) or, dually, find a minimal set of unsatisfiable clauses (the unsat core)
- New in Yices 1.0: It's all new

Responding to Changes in the World Around Us

Direct competitors; mysteriously, these still have users

PVS: ACL2, Caveat, Coq, HOL, Isabelle

SAL: NuSMV
(JPF, Spin, TLC etc. are explicit state—a different category)

Yices: Barcelogic, CVCL, MathSAT,

The new world; specialization

Static analyzers: Astrée, Absint, Coverity, ESC, Findbugs, Fortify, Prefix/Prefast, Smallfoot, Terminator, TLV

Dynamic analyzers: Daikon

Software model checkers: Blast, CMBC, SDV

Hardware model checkers: Cadence SMV and other CEGARs

Timed and hybrid systems: Charon, Checkmate, UPPAAL

And real languages: C, Simulink/Stateflow, Verilog (UML?)

Responding to These Changes

- Cannot be a single “one size fits all” solution
- The big systems must be open to customization and extension
- And we need an integrating environment for tying big systems and (many) specialized components together

Open to Customization and Extension

- We've changed PVS, SAL and Yices to "dual license" models
- PVS and SAL are now Open Source under GPL
  - Have made all the changes necessary for compliance
  - Ported PVS to CMU Lisp (Allegro still available)
- Yices is "free binary," GPL in the future
- Result: binaries for all our systems can be downloaded without hassle, commercial use is OK
- Dual license: `http://en.wikipedia.org/wiki/Dual_license`
- We own the copyright on all these, so can create custom licenses for commercial developers
- Consequence: contributors to the main branches are asked to assign copyright to us (there's a form for this)

Open to Customization and Extension (ctd.)

There have been many valuable contributions to PVS

- Libraries (NASA, Jerry James, many others)
- Strategies (Myla Archer, Hanne Gottliebsen, César Muñoz, several others)
- PVSio, ProofLite, Besc (César Muñoz)

And integrations with other systems

- Computer algebra systems (Hanne Gottliebsen and others)
- Other specification systems (Chris George and others)

Hope that open source encourages more of these, and creates opportunities for new kinds of contributions

We're also extending our web pages with Wikis for each system

Hope for collaborative FAQs, documentation, examples etc.

Open to Customization and Extension (ctd. 2)

We'd love to see projects extending these systems; we'll host and help student visitors if you can pay them; examples

- PVS ported to SCH
- PVS evaluator for Ada, C
- PVS random tester exploiting Yices
- Proof strategy for existential formulas that uses testers
- Platform-independent GUI for SAL (50% done)
- GUI/visualizer for SAL module compositions
- Counterexample visualizer
- Incremental BMC that retains SAT/SMT context
- Explicit-state model checker for SAL (90% done)
- Predicate abstractor using Yices
- Exploitation of multi-core CPUs

An Integrating Environment

- We have a proposal for an Evidential Tool Bus
  - The topic of my talk at Verify'06 last week
  - `http://www.csl.sri.com/~rushby/abstracts/sefm06`
- An implicit invocation blackboard architecture that uses two dimensions of logic (sublogic/representation) as its ontology
- Expect to start work next month
- Plan to attach our tools and components, and those of Blast
- And invariant generators based on predicate abstraction and static analysis (Ashish Tiwari's logical lattices)
- Hope to reconstruct analyzer for Hybrid SAL based on hybrid abstraction, and a model checker for C

An Integrating Environment (ctd. 2)

- We'd love to see projects helping develop the ETB, adding components, and using it
- We hope to make it an integral contribution to the Verified Software Grand Challenge
- Examples
  - Reconstruct LAST
  - Build an ESC for C
  - Integrate a dynamic analyzer (e.g., Daikon)
  - Integrate visibly pushdown automata
  - Do controller synthesis
  - Do a cool application

Summary

- We hope you find these licensing changes and our technical plans attractive and stimulating
- And that you'll contribute to the development, use, documentation, and support of the tools
- And, above all, help us expand this community
- We welcome your suggestions and comments