Connecting Tools Together

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA
Horses for Courses

- No one formal methods tool is universally effective
  - Often want to apply several tools to a single design description
  - Tool suites are usually of uneven quality
- Results from one tool can help further analysis by another
- The components of one tool may be useful to another
Apply Several Tools To A Single Description

- Cf. IF, Möbius, Veritech,…

- Our approach, SAL, is based on transition relations (nondeterministic state machines) as the common representation

- Natural representation for several kinds of computational systems

- And used by many Off-The-Shelf tools
SAL Language

- A way to specify transition relations and their properties
- Developed in a loose collaboration with Stanford (David Dill), Berkeley (Tom Henzinger), and Verimag (Saddek Bensalem)
  - InVeSt has adopted the SAL language
- Has definitions, guarded commands, modules, synchronous and asynchronous composition
- External representations use XML
SAL Tools

We provide

- Parser and prettyprinter
- Typechecker
  Like PVS, can use theorem proving to discharge certain “well-formedness” requirements
    - E.g., no causal loops in synchronous composition
- Predicate and data abstractor
- Explicit-state model-checker
- Tool Bus

Users provide

- Translators from front-end notations into SAL
- Wrappers/translators from SAL to back-end tools
SAL Architecture

The SAL Language serves as a hub

Verilog → SAL
UML → SAL
Java → SAL

SVC → SAL
Murphi → SAL
Mocha → SAL
PVS → SAL
SPIN → SAL

Programs
Verification conditions
Abstractions
Properties

Not all of these boxes are populated yet

John Rushby, SRI
Automated Abstraction

- Calculate simplified system description that (hopefully) preserves the property of interest (cf. Predicate abstraction, abstract interpretation)
- The calculation is done by automated theorem proving
- The general theorem proving problem is undecidable
  - Full automation requires heuristics, which sometimes fail
- Classical verif’n poses correctness as single “big theorem”
  - So failure to prove it (when true) is catastrophic
- Abstraction creates a context for failure-tolerant theorem proving
  - Prove lots of small theorems instead of one big one
  - In a context where some failures can be tolerated
Integrated, Iterated Analysis

Results from one tool can help further analysis by another

Example

- Abstraction can work better if you know invariants
- A model checker can calculate the reachable states (strongest invariant) of a finite state system
- Concretization of the reachable states of an abstraction is an invariant of the original
- So calculate crude finite state abstraction, generate invariant with model checker, concretize, and iterate
- Final verification by model checking accurate, simple model
Even More Integrated, Iterated Analysis!

- (Approximations to) fixpoints of weakest preconditions or strongest postconditions also generate invariants and can strengthen those extracted from an abstraction
  - Mechanized by theorem proving
  - (Strongest postconditions are equivalent to symbolic simulation, which is independently useful)

- Counterexamples from failed model check help distinguish bugs from weak abstractions, and also help refine the abstraction
  - Suggest additional properties (invariants) that will help the theorem prover construct a tighter model
  - Suggest additional predicates on which to abstract
Truly Integrated, Iterated Analysis!

- Recast the goal as one of calculating and accumulating properties about a design (symbolic analysis)
- Rather than just verifying or refuting a specific property
- Properties convey information and insight, and provide leverage to construct new abstractions
  - And hence more properties
- Requires restructuring of verification tools
  - So that many work together
  - And so that they return symbolic values and properties rather than just yes/no results of verifications
- This is what SAL is about: Symbolic Analysis Laboratory
The Components Of One Tool May Be Useful To Others

- Lots of people use PVS just to get at its decision procedures
- We are making new faster version of these available as ICS
  - ICS = Integrated Canonizer-Solver (≡ ICanSolve)
- Decides combination of: propositional satisfiability, equality over uninterpreted function symbols with (linear) arithmetic, arrays, datatypes
- Will later extend to quantifier elimination for decidable fragment of these
- Differs from other packaged decision procedures (e.g., SVC) in having rich API for adding/retracting facts, testing formulas
- Aim is to enable invisible or ubiquitous formal methods

John Rushby, SRI
What We Are building

SAL

PVS

ICS
Invisible Formal Methods

Use the power of automated deduction, abstraction, and model checking to augment traditional tools

- Extended static checking (cf. Compaq SRC’s ESC)
- Table checkers (cf. Ontario Hydro)
- Statechart/Stateflow property checkers (cf. OFIS)
- Test case generators (cf. Verimag/IRISA TGV)

And much more to come...
From Refutation To Verification

Assurance for system

- typechecking
- static analysis
- refutation
- automated abstraction
- experimental tools
- SAL
- InVeSt
- theorem provers
- verification

Effort

John Rushby, SRI

Invisible Formal Methods: 15
Acknowledgments

- N. Shankar, Sam Owre, Harald Rueß, Hassen Saïdi

- Saddek Bensalem, Jean-Christophe Filliâtre, Klaus Havelund, Friedrich von Henke, Yassine Lakhnech, César Muñoz, Holger Pfeifer, Vlad Rusu, Eli Singerman, and many others