NASA Langley Formal Methods Workshop; 13–16 June 2000
Williamsburg VA

An Overview of SAL

Saddek Bensalem, Vijay Ganesh, Yassine Lakhnech, César Muñoz, Sam Owre, Harald Rueß, John Rushby, Vlad Rusu, Hassen Saïdi, N. Shankar, Eli Singerman, and Ashish Tiwari

Computer Science Laboratory
SRI International
Menlo Park CA USA

And Verimag, Stanford, ICASE, IRISA Rennes, SRI SDL, Intel Haifa, SUNY-SB

Historical Context

- SAL is next in a succession of verification systems (deductively-based systems to support mechanized formal methods) developed at SRI: HDM, (STP), Ehdm, PVS, (InVeSt),. . .
- Each was more powerful, more cost-effective, and more widely-used than its predecessor
- Hope to achieve the same for SAL
- And maybe even to "get off the curve"
  (at least, for concurrent systems)
- PVS is a component of SAL

Back to Basics: Mathematics in Engineering

- Build mathematical models of a design, its environment, and requirements
- Use calculation to establish that the design in the context of the environment satisfies the requirements
- Only useful when mechanized (cf. CFD)
- Used in the design loop (exploration, debugging) as well as for verification
- Need to be sure that design is implemented correctly, and that environment and requirements models are "correct"
    - These are separate concerns

Formal Methods

- Same idea
- The applied math of Computer Science is formal logic
- So the models are formal specifications in some logical system: $\textit{environment} + \textit{design} \vdash \textit{requirements}$
- And calculation is mechanized by automated deduction: theorem proving, model checking, etc.
    - Undecidable in general; need human guidance
    - Decidable problems have much greater computational complexity than mechanizing continuous mathematics
- Must often trade off accuracy or completeness of the model for tractability and automation of calculation
    - E.g., downscale with a machete to allow model checking

Formal Methods for Assurance and Certification

- Formal calculations can cover all modeled behaviors, even if numerous or infinite (the power of symbolic reasoning)
  - Whereas complete testing is infeasible
  - And incomplete testing is unreliable
- Want highly accurate models, so that formal calculations support strong claims
- Then, using formal calculations, some activities that are traditionally performed by reviews
  - Processes that depend on human judgment and consensus

  can be replaced or supplemented by analyses
  - Processes that can be repeated and checked by others, and potentially so by machine
- Language from DO-178B/ED-12B

Adoption of Formal Methods for Assurance

- Must use accurate models
- Whose analysis generally requires interactive theorem proving
  - Automatic (heuristic) theorem proving seldom succeeds
    (so human guidance is provided indirectly)
- Driving a theorem prover is a specialized skill
- Often more about the prover than the design being examined
  - (Most) engineers will not do it
- FM may provide improved assurance, but at additional cost
  (unless there is regulatory relief on other processes)
- Provides few other benefits (so not used in the design loop)
  - Difficult to distinguish causes of failure:
    flawed proof, or flawed design?
- Overall: a difficult sale

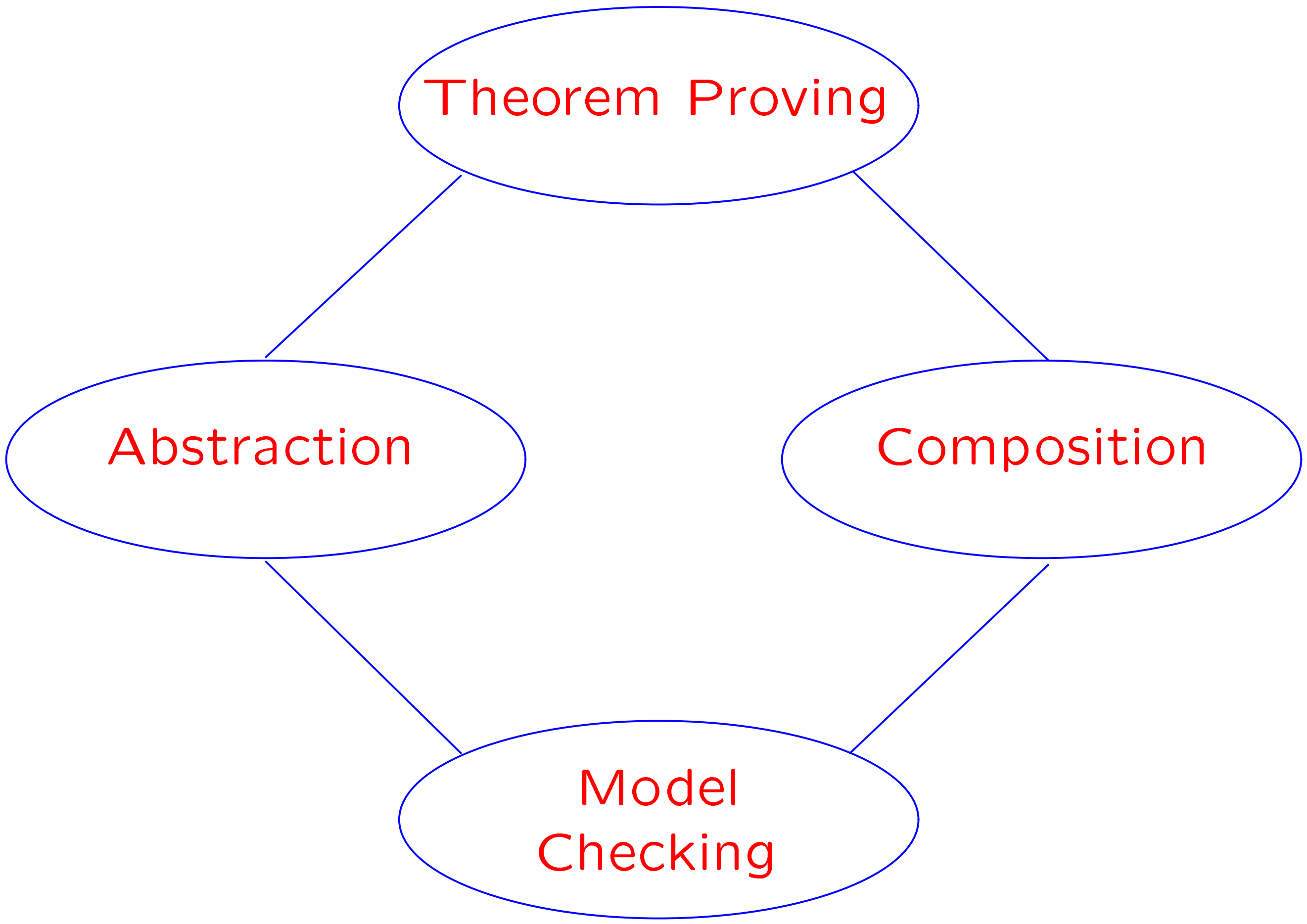
Adoption of Formal Methods Elsewhere

- Formal methods are quite widely adopted in certain industries (hardware, telecoms) for debugging (and design exploration)
- When the goal is debugging
  - Can use highly simplified or approximate models
  - So that highly automatic methods become available
    - ⋆ E.g., finite-state model checking
    - ⋆ Which also provides counterexamples
  - Interaction is through construction of the simplified models—this is like programming or design, and engineers don't mind doing it
- Large benefits in cost, reliability, and time to market
- But with simplified models, the failure to detect bugs does certify their absence—so no use for assurance

Overcoming Barriers to Adoption in Certification

- Methods must work for both debugging and assurance
  - Debugging provides the cost benefit
- Must be automated
  - Use model checking where possible
  - Theorem proving must be automatic
    - ⋆ Heavy use of decision procedures, quantifier elimination
    - ⋆ But also need heuristics, which will fail sometimes
    - ⋆ So need a framework in which a failed proof is not catastrophic
- Interaction must be through modeling
- Correctness-preserving abstraction is the key

Abstraction is the Bridge Between Deductive and Algorithmic Methods



Automated Abstraction

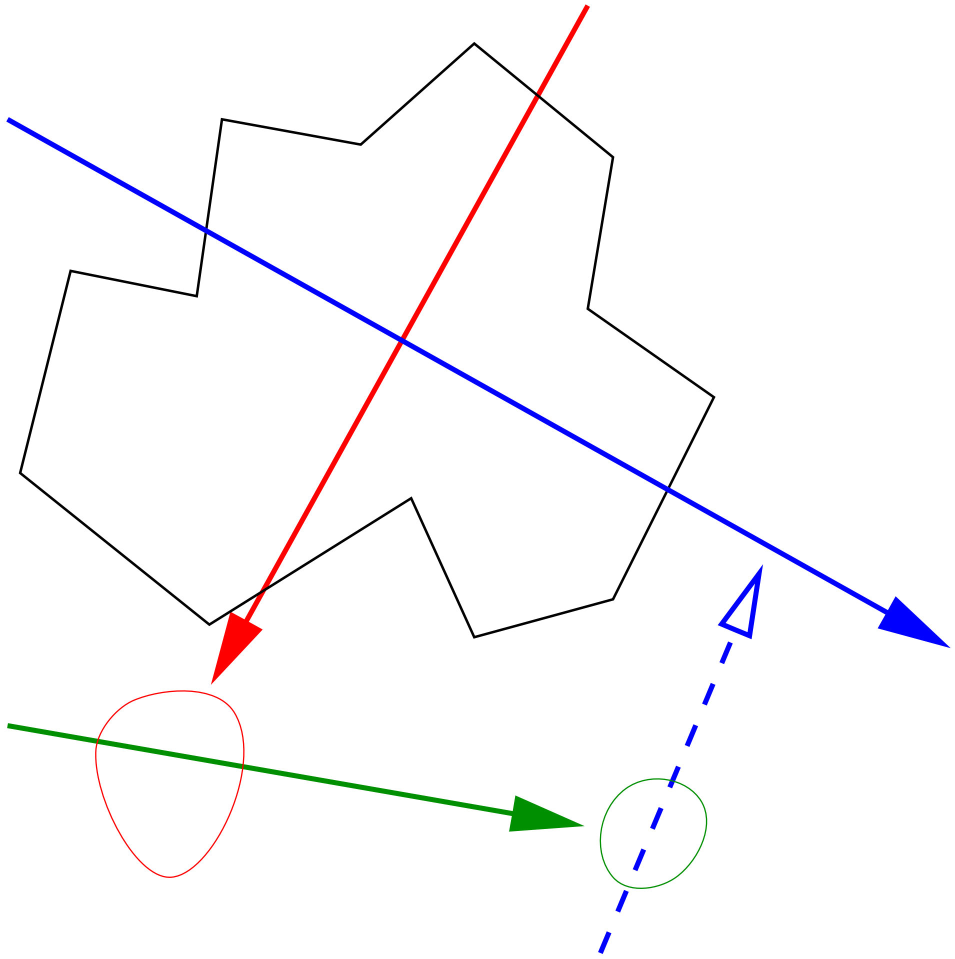
- Can often construct a simplified model that is faithful to the original (for a given property of interest)
  - The reduced model can by analyzed by model checking
  - And failure to detect bugs does certify their absence
- These reduced models can be constructed automatically by mechanized predicate abstraction (user suggests predicates)
  - The construction is done by trying to prove lots of little theorems (verification conditions)
    - ⋆ If a proof fails, reduced model will be more conservative, but is often still good enough
  - Already available in PVS as (abstract-and-mc)
- But the construction often requires auxiliary invariants

The Bridge Goes In Both Directions

- Model checkers often calculate the reachable stateset
  - Which is the strongest invariant

  And then throw it away
- The concretization of the reachable states of an abstraction is an invariant of the concrete system
  - And often a strong one
- So modify a model checker to return the reachable states as a formula that a theorem prover can manipulate
- Has been done (by Sergey Berezin) for CMU SMV and is used in InVeSt (another component of SAL)

Integrated, Iterated Analysis



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!

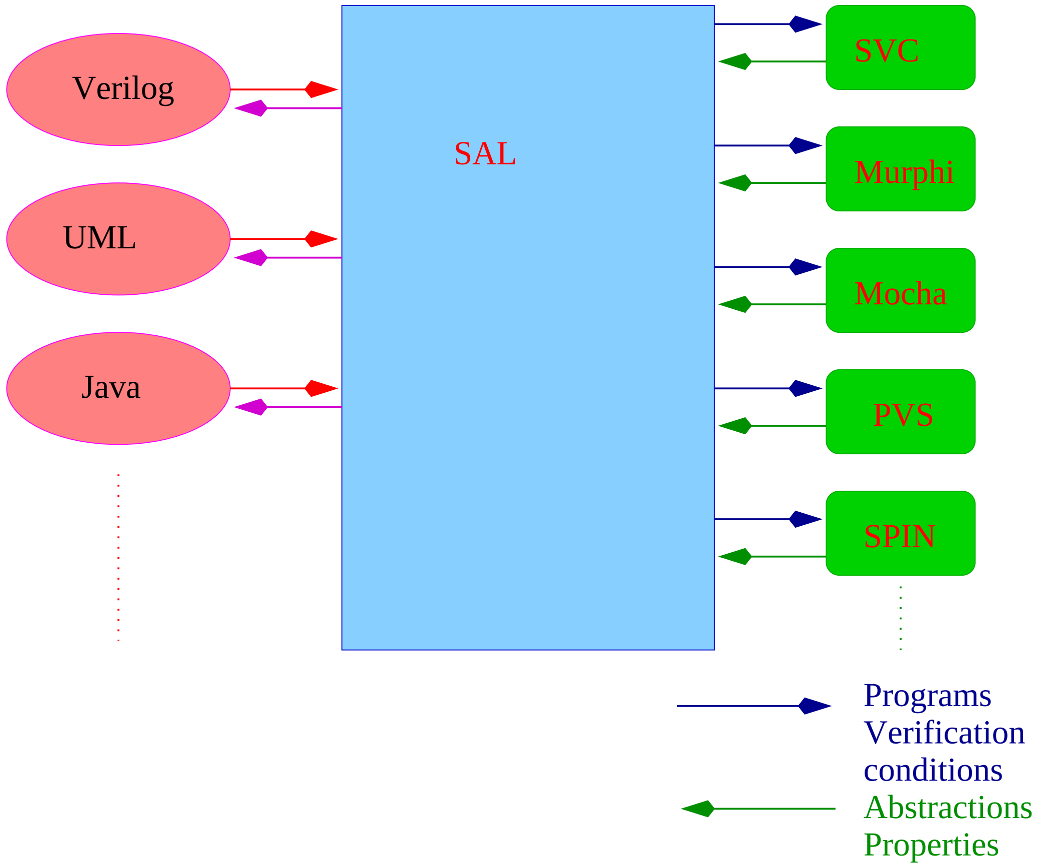
- Recast the goal as one of calculating and accumulating properties about a design (symbolic analysis)
- Rather than just verifying or refuting a specific property
- Properties convey information and insight, and provide leverage to construct new abstractions
  - And hence more properties
- Requires 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

Aside: Related Approaches

- Symbolic evaluation has some of the same benefits as integrated abstraction
  - Results of symbolic evaluation are considered in reviews (rather than have reviewers mentally execute the code)
  - Theorem proving is used to simplify expressions during evaluation
    - ⋆ Failure is not catastrophic
  - Used by Rockwell Collins for AAMP and JEM avionics and Java processors
- Domain-specific methods also seem viable (e.g., checkers for tabular specifications—lots of verification conditions) but these generally establish only weak properties
- PVS is being enhanced to do these better

SAL Architecture

The SAL Language serves as a hub



Not all of these boxes are populated yet

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)
  - Still converging: slightly different versions at SRI and Stanford
  - InVeSt has adopted the SAL language
- Has definitions, guarded commands, modules, synchronous and asynchronous composition

Current Translators (Prototypes)

- SAL to PVS
- SAL to SMV
  - Validated by César Muñoz at ICASE on "autopilot" example (SPIN 99): as efficient as hand-written SMV
- SAL to Java (for animation)
- Verilog to SAL (under development by Vijay Ganesh)
- All implemented in Java: currently being reconfigured to drive all translators off a common grammar and abstract syntax

Other Components

- SAL typechecker nearing completion
  - Like PVS, uses theorem proving to discharge some extended typecheck conditions (e.g., absence of causal loops)
- Concretization of counterexamples and reachable statesets returned from SMV is being prototyped
- And experimental invariant generation/strengthening
  - Can do 2-process bakery automatically
- Efficient predicate abstraction available in PVS 2.3
- Goal is to have a coherent system available for distribution this year

Reading, Effectiveness

- Predicate abstraction in PVS
  - Abstract and Model Check While You Prove, by Hassen Saïdi and N. Shankar, CAV '99 (LNCS 1633)
  - A tutorial description using Bakery in PVS at `http://www.csl.sri.com/~rushby/movep2k.html`
  - Solves BRP automatically in a minute or so
- InVeSt (uses LCS, MONA)
  - Also solves BRP automatically in a minute or so, CAV 98, CAV 99

Related Systems

- David Dill's treatment
  - Variant on predicate abstraction; can do garbage collector automatically in an hour or so, CAV '99
- Ken McMillan's treatment
  - Lightweight theorem proving whose subgoals are discharged by model checking; can do Tomosulo automatically, CAV 00
- Bandera (Matt Dwyer, John Hatcliffe, Kansas State)
  - Uses abstract intepretation, works on Java, ICSE 00
- SLAM (Tom Ball, Sriram Rajamani, Microsoft Research)
  - Boolean programs, WAVE 00
- Veritech (Shmuel Katz, Orna Grumberg, The Technion)
  - Another intermediate rep'n; eclectic front and backends

Summary

Goals and mechanisms of SAL

- Serve the needs of both verification and refutation
- Analyses must be almost completely automatic
- With interactive guidance expressed in terms of the model, its predicates, and its properties
- Use iterated, integrated application of abstraction, invariant generation, theorem proving, model checking
  - Model checking good for global properties
  - Theorem proving good for propagating local properties
- Using (modified) off the shelf tools where possible
- Linked through the SAL intermediate language

Prospects for Formal methods in Certification

- Mechanized formal methods are feasible (and used today) for simple designs (e.g., nuclear shutdown), and for implementations (cf. B, SACRES)
- But the challenge is complex designs (e.g., fault tolerant, partitioning, IMA architectures)
- Though non-critical but "sensitive" systems could become a technology driver (cf. E-commerce)
- Combined methods based on automated abstraction seem the best (the only?) hope
  - Traditional (labor intensive) methods will be outpaced by the growth in critical systems
- Tool qualification will be an issue
  - Need independently checkable products (proof objects)

Excitement Grows As The Model Checker Is Applied To The Final Abstraction