Darpa FM kickoff meeting, Seattle June 3,4 1998

Symbolic Analysis Laboratory (SAL)—Part 1

Joint project with David Dill, Stanford

John Rushby (representing Shankar)

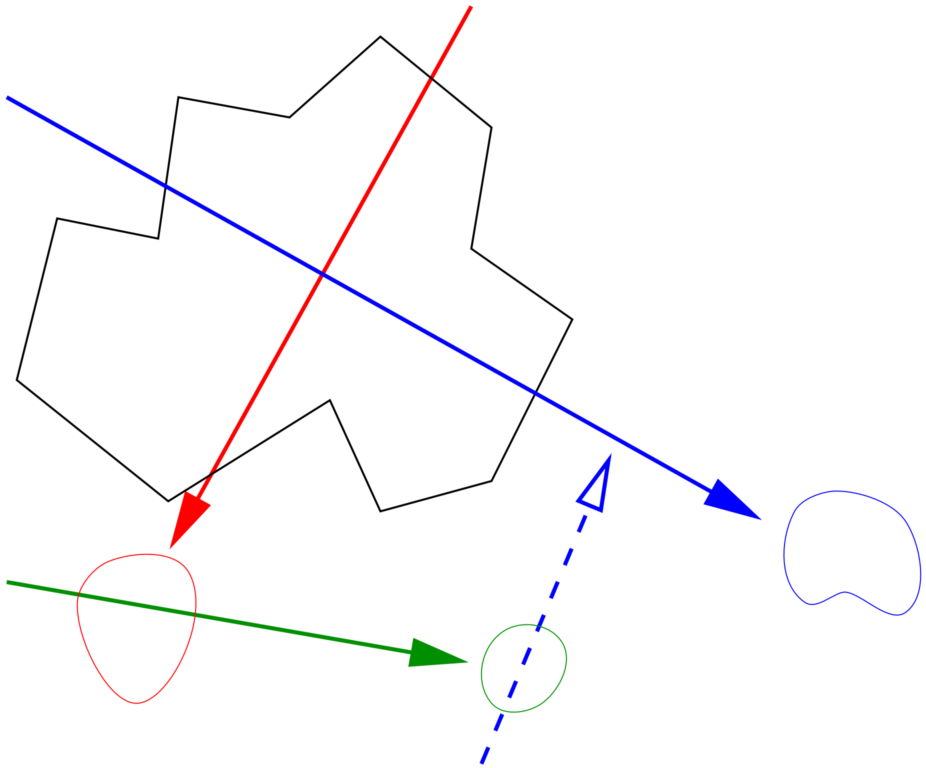
Computer Science Laboratory
SRI International
Menlo Park, CA

Summary

- Formal analysis of large or intricate problems is still a formidable task
- We can build ever more powerful tools, or we can make the problems easier
- We've worked on the first approach
  - And have had some success
- Now we're going to try the second
- Will do it by ubiquitous use of automated abstraction
  - That is, construction of simpler descriptions that ignore/approximate aspects of the original
- Within a framework that allows tools to cooperate through the exchange of symbolic values

Ubiquitous Automated Abstraction

- Generate models appropriate to different analyses and different tools from a single description
- Abstraction can be iterated
- Results from one analysis can assist further abstractions



Symbolic Values

- Many tools compute intermediate values that could be useful to other analyses
  - The set of reachable states
  - The symbolic state of a processor
- But throw them away and return only the true/false result
- We will retain these intermediate symbolic results and allow them to be passed to other tools
  - For example, the reachable states of an abstraction often provide a useful invariant of the concrete system
- Provides path for effective integration of multiple tools

Background

- This project continues a collaboration between SRI and Stanford that began over five years ago
- Individual starting points:

  SRI: Theorem proving approaches using PVS

  - Classical higher-order logic with enriched type system
  - Integrated environment with interactive theorem prover and powerful decision procedures
  - Many hundreds of users worldwide
    - ⋆ 90+ papers by non-SRI authors, 80+ by us

  Stanford: Model checking approaches using Murϕ

  - Dave will elaborate
- Formal methods provide design insight and assurance through calculation

More Background

- First joint project: developed tools and methodologies for using theorem proving and model checking in combination
  - Murϕ to PVS translator
    - ⋆ Downscale and debug in Murϕ, verify full thing in PVS
  - Decision procedure for Park's μ-calculus provides CTL model-checking within PVS
    - ⋆ Allows integrated theorem proving and model checking
- Second joint project: a federated environment for development of customized formal methods tools
  - Reusable components + glue
  - Canceled after less than a year!
  - Valuable assets developed and used (Dave will describe)
    - ⋆ Needs to be revived

Experience—Good

Cache coherence, for example (Dave will elaborate)

- Formalization in Murϕ used in validating Sparc V9 memory model
- Murϕ used in design loop of commercial cache-controllers
- Systematic method ("aggregation") allows straightforward verification of full protocols (e.g., Flash)

But still needs substantial human insight and construction

Experience—Less Good

Bounded Retransmission Protocol (BRP), for example

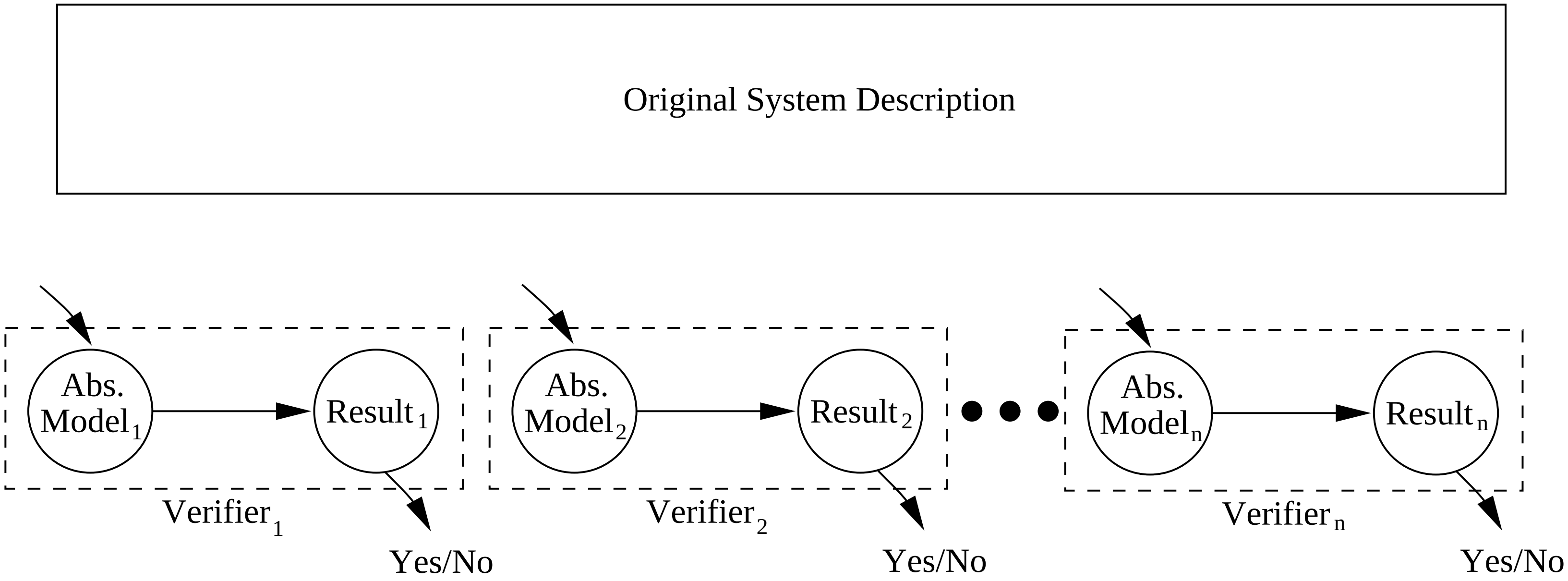
- Classical invariance proof
  - Desired invariant is not inductive: needs strengthening with additional conjuncts
    - ⋆ Plausible conjuncts found by examining failed proof state
    - ⋆ Checked with Murϕ
    - ⋆ Added to invariant (without quitting proof)
  - Iterated the above—57 times!
- Also tried verified abstraction plus model-checking
  - But needed 45 invariants to establish abstraction

Moving Ahead

- Many problems of practical interest are beyond the reach of current theorem-proving approaches
  - Problem is not deductive power, but ancillary tasks such as invariant generation
- And model checking has scaling problems, too
  - Have to construct downscaled or abstracted models
  - Need different tools and models for different purposes
  - Must deal with the state explosion problem
- The SAL ideas (Symbolic Analysis Laboratory)
  - Automate the abstraction and invariant generation
  - Allow tools interoperate by exchanging symbolic values rather than yes/no results
- Project has not started yet

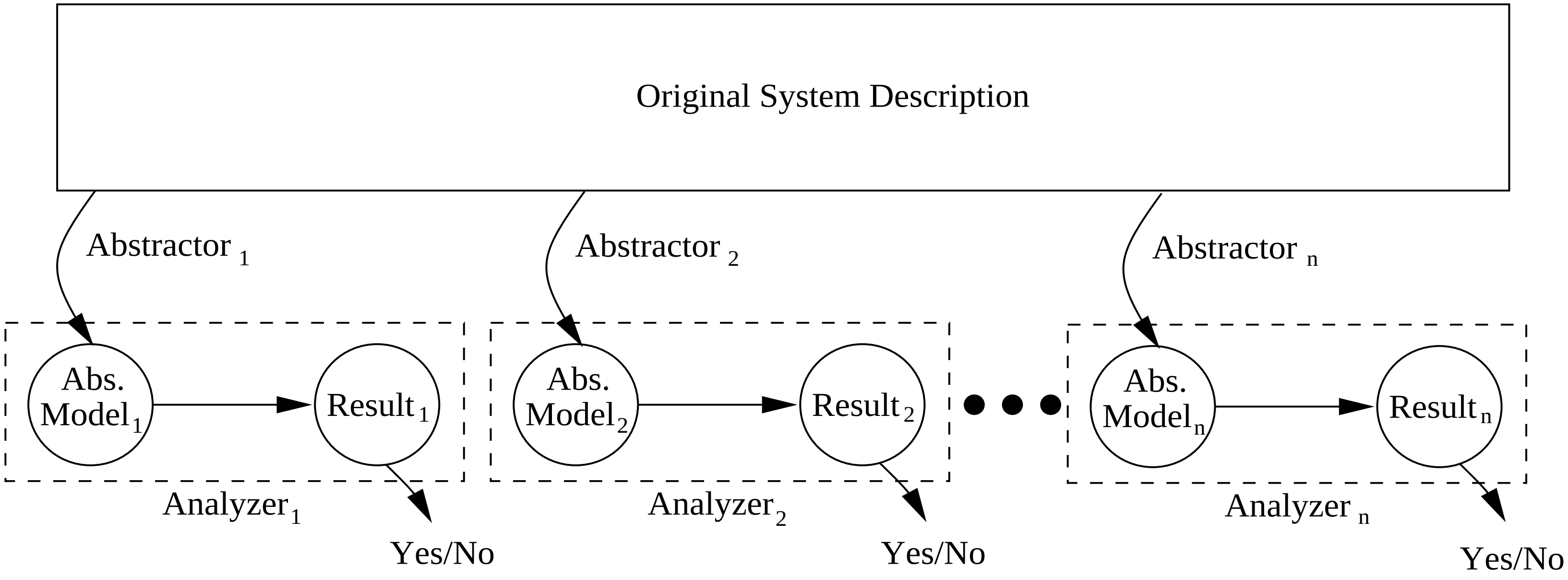
Formal Analysis Today

- The abstracted models are built by hand
- Only results are yes/no verifications (and, possibly, counterexamples at the level of the abstracted description)
- No connection between different tools, nor between different methods of analysis



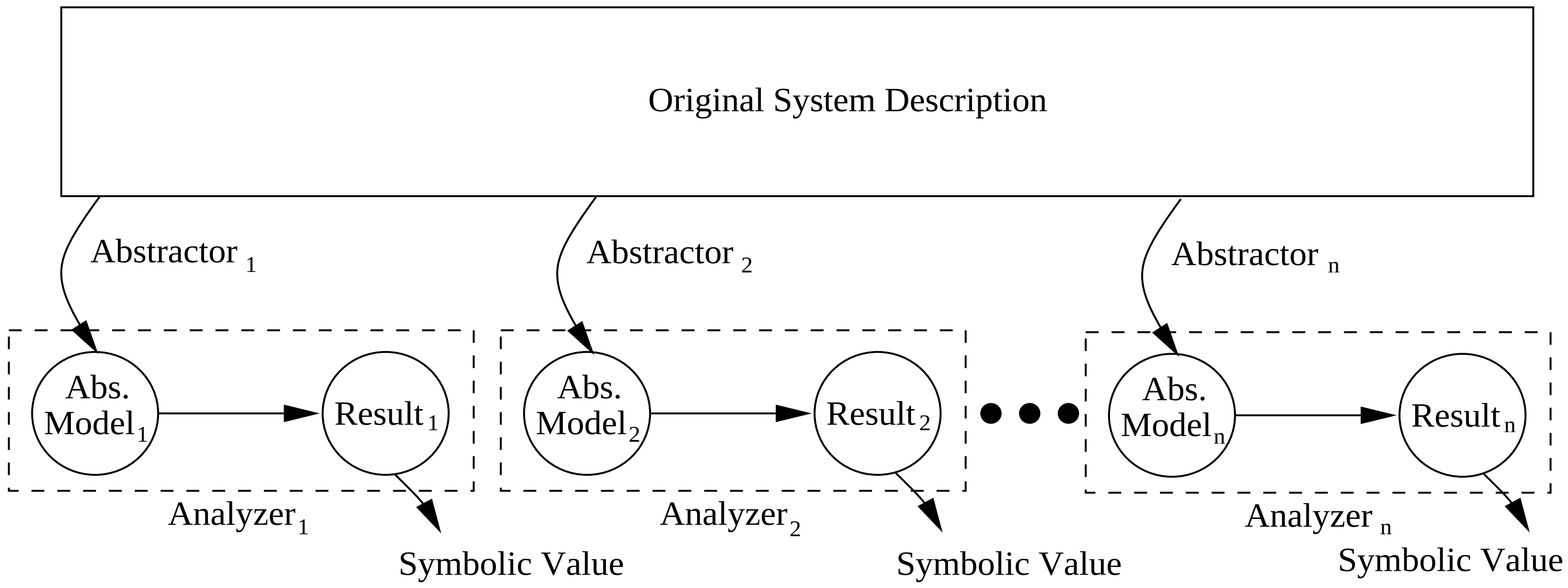
The SAL Vision—Part I

- The models are constructed by automated abstractors
- Different abstractions for different tools are constructed from a single description



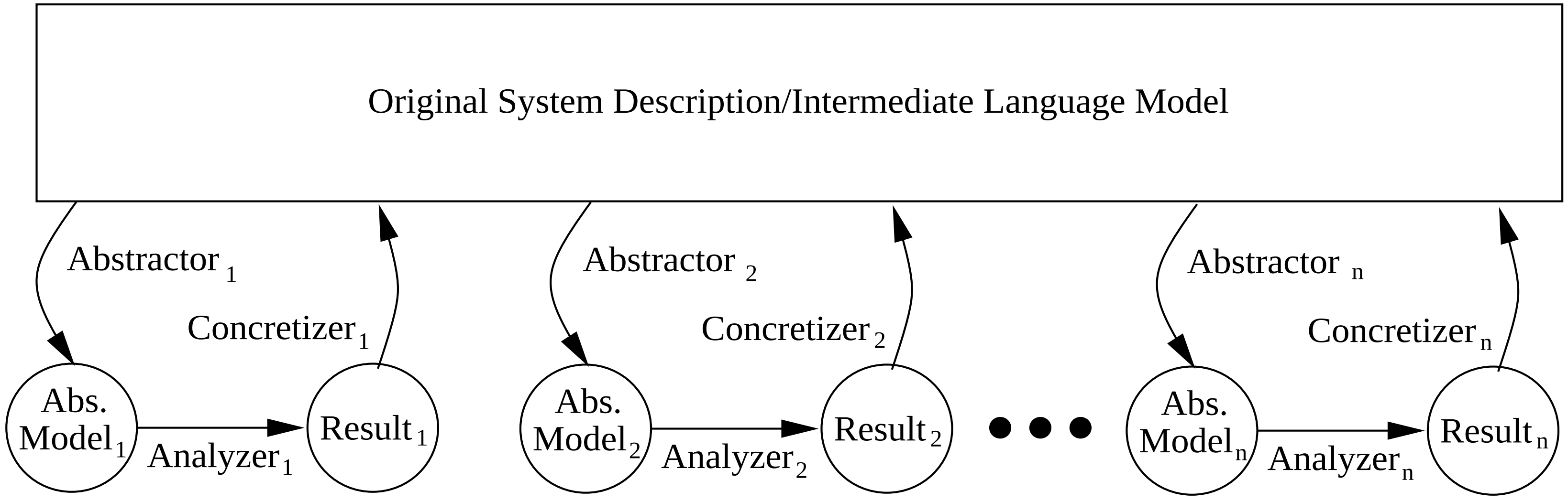
The SAL Vision—Part IIa

- The tools produce symbolic values (e.g., a set of reachable states, or the symbolic state of a processor) that can assist further analyses



The SAL Vision—Part IIb

- Symbolic values are converted back into the terms of the original description by automated concretizers
- And the whole process can iterate through several tools, each using results computed by others



Abstractors

- Idea is to automate some or all of 2, 3, 4 below
  1. Given concrete system
  2. Develop abstracted system
  3. Relate these by an abstraction function
  4. Show that the abstraction is sound
  5. Show that the abstraction has the property of interest
- And arrange things so that 5. is easy (e.g., finite state)
  - iterate if not
- Two broad classes (can be seen as variants of each other)
  - Syntax-directed (cf. abstract interpretation, slicing)
  - Semantic (i.e., their construction requires deduction)
- Unsound abstractions can be useful for debugging (cf. downscaling)

Current Activities

InVeSt: adjunct to PVS

- Uses automated deduction + syntax-directed substitutions to calculate abstracted system descriptions

Graf-Saïdi: originally developed at Verimag as adjunct to PVS

- Uses predicate abstraction, with static analysis + model checking to compute invariants
- Dave will describe further developments at Stanford

SAL Language: design of the intermediate language

- To serve as source and target for abstracted system descriptions

InVeSt—I

- Developed at SRI by Saddek Bensalem (Verimag) and Yassine Lakhnech (Kiel) and Sam Owre
- Theory and system description to be presented at CAV
- User provides concrete system description (in stylized PVS)
  - Together with abstracted state space
  - And abstraction function
  - Built-in heuristics can often generate these automatically
- System computes abstracted system that is correct (conservative) by construction
  - Outputs it in either PVS or SMV
- Abstracted system has same structure as original
- Procedure can be iterated (i.e., abstract the abstraction)

InVeSt—II

- System described as a transition relation
  - `(pc, guard, updates, next pc)`
- Syntactic substitutions
  - Substitute updates in abstraction function and check if each abstract variable is suitably determined
- Elimination method (for remaining abstract variables)
  - Variables can be grouped for efficiency
  - Connect all relevant abstract states by a transition
  - For each such abstract transition
    - ⋆ Calculate verification condition that allows its removal
    - ⋆ Attempt automated proof of the verification condition
    - ⋆ Proof can use automatically-generated invariants

InVeSt Examples

BRP

- Use heuristically-generated abstraction function + one additional abstraction
- Finite state abstract system generated in 1 hr 20 mins

Cache-Coherence (simple snoopy cache)

- User proposes abstraction similar to Fong Pong's
- Finite state abstraction computed automatically

Alternating bit protocol

- User proposes the abstraction of Müller and Nipkow
- Finite state abstract system computed automatically
- Requires automatically-generated invariants, plus some lemmas on lists

Bakery, Peterson etc. reduced to finite state automatically

Looking Forward: Potential Impact

- Systems of all kinds are becoming highly integrated
  - Old style: design each "axis" in isolation
    - ⋆ Roll, pitch, yaw
    - ⋆ Engines, flight-control, weapons, sensors
    - ⋆ Real time, fault tolerance, security
  - New style: everything interacts with everything else
    - ⋆ Greater performance, also unanticipated interactions
    - ⋆ Much increased design, integration, and validation costs
    - ⋆ All-up simulation is the only validation method
- Automated abstractions could project out "single axis" views
  - Calculate individual properties
  - And interactions
  - Would provide a "science of integration"

To Learn More About PVS

- Browse general papers and technical reports at `http://www.csl.sri.com/fm.html`
  - `tse95.html` is a good overview of PVS
  - `pvs-bib.html` links to a bibliography with over 180 entries
- Detailed Information about PVS is available from `http://www.csl.sri.com/pvs.html`
  - `http://www.csl.sri.com/pvs/examples` gets you to a directory of tutorials and supporting specification files
- You can get PVS from `ftp://ftp.csl.sri.com/pub/pvs`
  - Allegro Lisp for SunOS, Solaris, IBM AIX, or Linux
  - Need 64M memory, 100M swap space, Sparc 20 or better (Best is a high-end Intel machine, costs under $2,000)