

SAL 2

Leonardo de Moura

(joint work with Sam Owre, Harald Rueß, John Rushby,
N. Shankar, Maria Sorea, Ashish Tiwari)

`demoura@csl.sri.com`

Computer Science Laboratory

SRI International

Menlo Park, CA

Symbolic Analysis Laboratory

- ▶ Set of tools for analysis of state machines
- ▶ Symbolic model checkers (LTL and CTL)
- ▶ Bounded model checker
- ▶ **Infinite** bounded model checker (reals and integers)
- ▶ Simulator
- ▶ k -induction
- ▶ Finite & Infinite systems
- ▶ Scriptable (Scheme)

The Language

- ▶ It was originally conceived as an **intermediate language**.
- ▶ Developed in collaboration with the research groups of:
 - ▶ David Dill (Stanford)
 - ▶ Tom Henzinger (UC Berkeley)
- ▶ The language has evolved and it is now a **comprehensive specification language**.
- ▶ The type and expression language are similar to those of PVS. For instance, it includes **quantification** and **recursive functions**.
- ▶ State machines are specified as parameterized modules with state variables explicitly identified as **input, output, local, or global**.

The Language (cont.)

- ▶ Modules may be **composed** both **synchronously** and **asynchronously** (and in combinations of these) to yield systems.
- ▶ A **renaming construction** allows inputs and outputs of different modules to be **“wired up”**.
- ▶ It is easy to translate most other state machine languages into SAL.
- ▶ We have a translator from the Stateflow notation of Matlab/Simulink.
- ▶ SAL has a PVS-like notation for humans, and fairly simple XML and Lisp-like input languages for other programs.

Preprocessing and Compilation

- ▶ All tools share a common set of preprocessing and compilation routines:
 - ▶ Partial evaluation
 - ▶ Common subexpression elimination
 - ▶ Slicing
- ▶ LTL assertions are translated to optimized Büchi automata.
- ▶ SAL 2.3 also provides a lightweight typechecker, called the SAL well-formedness checker.
- ▶ Deeper checks needed for some of the richer constructions: these require generation and proof of TCCs and will be supported by the full SAL typechecker, which is based on that of PVS.

Symbolic Model Checkers

- ▶ They use the **CUDD** BDD package.
- ▶ They deliver **performance** and **functionality** comparable to other **state-of-the-art** symbolic model checkers.
- ▶ The core functions are available in the **scripting API**.
- ▶ The **Witness Model Checker** (WMC) implements a novel approach that constructs both (positive) witness and (negative) counterexamples for CTL assertions.

Bounded Model Checker

- ▶ It uses a propositional **SAT solver** to search for counterexamples no longer than some specified “**depth**”.
- ▶ **Iterative Deepening**
- ▶ Verification using **k -induction** (optionally using other formulas as **lemmas**).
- ▶ Counterexamples for the k -induction step.
- ▶ Supported SAT solvers: **ICS**, **zChaff**, and **GRASP**.

Infinite Bounded Model Checker

- ▶ Similar to a standard bounded model checker.
- ▶ It translates to the theory supported by **ICS** instead of a purely propositional SAT problem.
- ▶ ICS is a decision procedure and satisfiability solver for the combination of ground (i.e., unquantified) real and integer linear arithmetic, equality with uninterpreted function symbols, products and co-products, propositional calculus, bitvectors, arrays, and etc.
- ▶ Using **real** or **unbounded integer** state types, SAL can represent infinite state systems such as **hybrid** or **timed automata**.
- ▶ Other supported solvers: **UCLID**, **SVC**, **CVC**, and **CVC-Lite**.

Scripting and the SAL simulator

- ▶ The preprocessing and model checking components of SAL can be accessed through an **API** defined in Scheme.
- ▶ The actual model checkers are simply Scheme scripts defined over this API.
- ▶ Users can write their own scripts to perform specialized analyses using the full resources of SAL.
- ▶ The **SAL Simulator** provides a convenient environment in which to develop such scripts (**read-eval-print** loop).
- ▶ Example: **test case generator** for Stateflow.

Applications

- ▶ Startup algorithm for TTA (time-triggered architecture)
- ▶ Hybrid SAL: Biology
- ▶ Priority Ceiling Protocol
- ▶ Needham-Schroeder Protocol
- ▶ Suzuki-Kasami Distributed Mutual Exclusion Algorithm

Plans for Further Development

- ▶ An **explicit-state model checker** (it was available in SAL 1.0).
- ▶ Integrate SAL with PVS.
- ▶ Predicate Abstraction Tool.
- ▶ More front-ends (translators).
- ▶ An open scriptable environment for symbolic analysis in which numerous tools (**SAL Tool Bus**).
- ▶ Evidence Management System.

Current Status and Availability

- ▶ SAL 2.3 is freely available for noncommercial research purposes from `sal.csl.sri.com`.
- ▶ **Binary versions** of the system, which require an automatically-generated license key, may be downloaded for: **Linux**, **Solaris**, **MacOS X**, and **Cygwin** (for Windows).
- ▶ The SAL and ICS source code is available with a signed license agreement.
- ▶ The top-level page for tools developed by our group is `fm.csl.sri.com`.