An Abundance of Tools for Automated and Semi-Automated Formal Reasoning

Assaf Kfoury

January 23, 2017 (last modified: December 26, 2017)

There is a profusion of software *packages* (or *tools*) that have been developed, and constantly improved, to assist computer scientists in the tasks of *formal* (or *logical*) reasoning. Each of these tools is designed for a specific purpose, *i.e.*, to assist in tackling a specific class of problems. Such a class of problems may be more "generic" than another, in the sense that it covers a wider area of potential applications, so that a tool designed to tackle them is necessarily less domain-specific and more general.

A tool for automated or semi-automated reasoning involves a *modeling language* (a formal language), suitable for expressing properties of a system (software or hardware) and for specifying its behavior, together with an *underlying logic* (a formal logic) to reason about system properties that are thus expressed.

Depending on the modeling language and underlying logic, these tools are variedly called *SAT solvers*, *SMT solvers*, *QBF solvers*, *model checkers*, *probabilistic model checkers*, *proof assistants*, *theorem provers*, *interactive theorem provers*, and other names that are suggestive of what they do. The demarcation between them is not sharp; *e.g.*, every SMT solver is already a SAT solver and nearly every model checker is built on top of a SAT solver. They can be broadly divided into three groups:

1. For a partial list of **SAT/SMT solvers** and what they do, click here.
   Other well-known SAT/SMT solvers in decreasing order of preference (from my own experience):

   Z3 (click here), CVC4 (click here), Yices (click here), and Alt-Ergo (click here).

   For a SAT solver, the modeling language and underlying formal logic is that of propositional logic. For a SMT solver, the modeling language and underlying formal logic is a fragment of first-order logic extending propositional logic.

2. For a partial list of **model checkers** and what they do, click here.
   Some of the better known model checkers are:[1]

   - SPIN (click here), "an open-source software verification tool, ... used for the formal verification of multi-threaded software applications",
   - PRISM (click here), "a probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour",
   - Alloy (click here), "a formal language for describing structures and a tool for exploring them".

   The modeling language of a model checker is often (not always) a customized version of an existing programming language. For example, the modeling language for SPIN is called Promela

[1]Or at least I am more familiar with them.

(click here), specially designed as a verification modeling language. The underlying logic of a model checker is typically a temporal logic (*e.g.*, LTL, CTL, etc.) or a modal logic (*e.g.*, μ-calculus, TLA, etc.). A model checker is often implemented on top of a SAT solver (*e.g.*, the Alloy tool works by reduction to a SAT solver).

3. For a partial list of **interactive theorem provers/proof assistants** and what they do – or what they are good at – click here. Among the best known in decreasing order of preference (from my own perspective), though none is best for all possible situations:

   - Coq (significant prerequisite: constructive dependent type theory) – click here,
     "Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs"
   - Isabelle/HOL (significant prerequisite: simple type theory) – click here,
     "Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus"
   - ACL2 (significant prerequisite: primitive recursive arithmetic) – click here,
     "ACL2 is a logic and programming language in which you can model computer systems, together with a tool to help you prove properties of those models"
   - PVS (significant prerequisite: classical dependent type theory) – click here,
     "PVS is a verification system: that is, a specification language integrated with support tools and a theorem prover. It is intended to capture the state-of-the-art in mechanized formal methods"

   The modeling language and underlying logic of an interactive theorem prover are typically those of a higher-order logic, which is most often (not always) typed and may involve powerful features such as *dependent types*.