HSMTLib
Interacting with SMT Solvers in Haskell

Nuno Laranjo, Rogério Pontes, and Maria João Frade

HASLab/INESC TEC & Universidade do Minho

Introduction. Satisfiability Modulo Theories (SMT) is the problem of deciding the satisfiability of logical formulas over one or more first-order theories. SMT has applications in diverse domains and is the subject of very active research. The progress achieved by SMT solvers in the last decade has enabled its effective use in many applications and made them pervasive in the context of software verification, including deductive program verification, static program analysis, run-time analysis, type inference, as well as model-based tools, test-case generation, and proof assistants. One should refer the normative effort of the SMT-LIB initiative associated with the competition SMT-COMP, to provide a common input and output format and benchmarking framework for the evaluation and comparison of SMT solvers. The SMT-LIB 2.0 format is the current standard interface for SMT solvers.

We have developed a Haskell library (HSMTLib) for easy interaction with multiple SMT solvers. Usually SMT solvers provide a native API, most commonly, for C, and there are some wrappers for other programming languages. That is not very convenient when one wants to deal with multiple solvers, for instance to use the solver that performs better with a specific problem/logic. The few existing Haskell libraries that do provide solver interaction (such as yices-painless and Z3 bindings) are solver specific. Some other packages only provide an AST for creating SMT-LIB commands and expressions, as for instance smtLib.

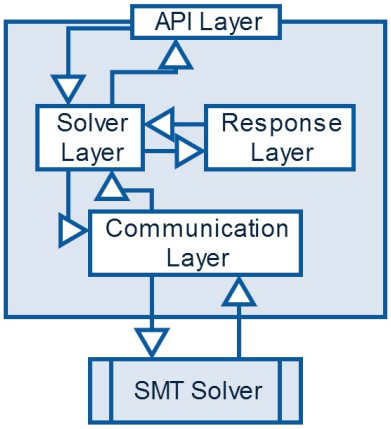
The HSMTLib library provides an API which can be used to work across all supported solvers. The library supports online interaction, allowing interactive communication with a solver, so that multiple queries to the solver can share common state. We think the HSMTLib library is useful to the Haskell community. Examples of Haskell programs that can benefit of its use include the SBV package, the Liquid-Haskell project or the Cryptol implementation.

HSMTLib description. HSMTLIb interacts with multiple SMT-LIB 2.0 compliant solvers and provides a common API which conforms to the SMT-LIB 2.0 standard. The solvers currently supported are CVC4, MathSAT and Z3, but we plan to add support for Yices, Boolector and Alt-Ergo within a short time.

The code in development is hosted at Github[1] and the last stable version can be downloaded from Hackage[2] or installed via Cabal. The library also has Haddock documentation available at Hackage and some additional tutorials are available in the Github Wiki[3].

The library provides two modes of interaction with the solvers, *online mode* and *script mode*. In online mode a solver is kept running and communication is done via pipes. In script mode commands are written to a script, when needed a solver is initialized with the script, and then the library awaits until the solver ends and returns the result.

[1] `https://github.com/MfesGA/Hsmtlib`
[2] `https://hackage.haskell.org/package/Hsmtlib`
[3] `https://github.com/MfesGA/Hsmtlib/wiki`



The library is divided in four parts: *API layer*, *solver layer*, *communication layer* and *response layer*. The API layer provides commands and most data types a developer must use to interact with the solvers. The solver layer is the core since it binds all the other layers: it implements the commands defined in the API layer by using the functions provided in the communication layer (depending on the solver in use), forwards the results to the response layer and finally returns the result obtained. The communication layer contains the functions to communicate with each solver. The response layer simply parses the results from the solver and turns it into a Haskell data type. The aim of this architecture is to facilitate the addition of new solvers.

For each solver supported there is a standard configuration that can be easily tuned when the solver is initialized. Let us give a very small example of a program which is using Z3 with the QF_IDL logic in online mode with the standard configuration:

```
main :: IO Result
main = do
    solver <- startSolver Z3 Online QF_IDL Nothing Nothing
    produceModels solver
    mapDeclConst solver ["x", "y"] tInt
    assert solver $ (ct "x"  `nAdd` ct "y") === lit 0
    assert solver $ (ct "x"  `nSub` ct "y") === lit 6
    sat <- checkSat solver
    case sat of
        CCS Sat ->  getValue solver [ct "x", ct "y"] >>= print
        CCS Unsat -> print "The constraint is unsatisfiable"
        ComError error -> print error
    exit solver
```

The option to produce models is activated. Then the integer constants x and y are declared, and the constraints $x + y = 0$ and $x - y = 6$ are asserted. Afterwards a satisfiability check is done and the answer from the solver is analyzed. If the constraints are satisfiable the command getValue is used to retrieve an interpretation given to x and y. Note that to change the SMT solver, for instance to CVC4, we just need to write CVC4 instead of Z3 in the first line.

The library was tested using two methods: by writing unit tests using HUnit; and by generating Haskell programs that run in online mode the benchmarks provided by SMT-LIB. More details about the library can be found in the technical report hosted at Github.

Future work. Using the HSMTLib we have stablished connection with several SMT solvers which claim to be SMT-LIB 2.0 compliant. Despite we found no problems in sending the requests to the solvers, we actually had some problems with some solvers because the answers given to some commands were not completely following the standard (or the standard is underspecified in some points). This is the reason currently the library only supports CVC4, MatSAT and Z3. However, we keep on improving the response layer in order to support more solvers. Moreover, for the next version of the library we plan to allow running multiple solvers concurrently.

Acknowledgment. This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within projects **FCOMP-01-0124-FEDER-020486** and **FCOMP-01-0124-FEDER-037281** .