Boolean Format Multitool for the Next Generation

Maximilian Heisinger
Johannes Kepler University, Linz, Austria, maximilian.heisinger@jku.at

Abstract

In parallel to the improvement of SAT and QBF solvers, problem *encodings* are also continuously refined. Formulas re-use frequently occurring building-blocks such as pseudo-Boolean- or cardinality constraints, which themselves are optimized ever more. However, since new problems are always authored as a whole, improvements of individual components do not benefit them until they are specifically integrated. We want to improve the encoding development process so that the advanced variants of commonly used constraints are already usable early-on, without adding too much friction to the encoding developer. Additionally, we want a converter for different formula formats that can be used as a basis for future tool and encoding research. We present *Booleguru*, an encoding and formula converter tool for future encoders.

1 Introduction and Project Goals

Booleguru's aim is to merge prior ideas and build a tool that is quick to pick up, while providing enough depth to develop encodings of more complex problems without running into scaling issues.

1.1 Limboole, the Boolean Calculator

Limboole [1] builds a custom language based on well-known rules of boolean expressions. The Limboole DSL syntacx for the boolean formula $a \wedge \neg(b \vee \neg c)$ is `a & !(b | !c)`. It uses a recursive-descent parser and then transforms the internal tree of logical operations into conjunctive normal form (CNF) with a direct Tseitin encoding. Our experience shows that a domain specific language (DSL) such as used by Limboole helps to encode problems quickly, especially when developers do not have much prior encoding experience.

1.2 PySAT and Encoding Modularity

PySAT [2] is a Python library without its own DSL. It wraps solvers and provides many different pseudo-Boolean (PB) and cardinality constraints. When working with PySAT, the user has a low-level view on their formulas. In contrast to Limboole, where one can directly join multiple formula parts together using logical connectives, one has to actively manage variable indices and clauses. Creating a higher-level interface for arbitrary boolean formulas is also stated as a long-term goal by the PySAT developers.

2 Accomplishing These Goals

Booleguru contains parsers, transformers and serializers. Parsers implement languages it can read, which is currently:

- an expanded Limboole-esque boolean calculator syntax with optionally embedded Lua or Fennel[1],
- DIMACS,
- and a reduced form of SMT-LIB2 [3].

The in-memory representation of the AST can then be transformed in multiple ways, e.g. distributed to CNF, Tseitin-encoded to CNF, or prenexed with different strategies. Finally, the resulting boolean expression can be printed out using a serializer (currently, a Limboole-esque format and QCIR are implemented). Booleguru can be used either through its API or its CLI. The latter is again a boolean calculator and can e.g. combine f_1 and f_2 into $f_1 \wedge \neg f_2$. An example invocation follows:

```
booleguru f1.dimacs --and --not f2.boole
```

When adding `--qcir`, the output is given in QCIR.

2.1 Current State and Future Development

While still early in development, the project looks promising and enables us to try out multiple ideas without having to re-implement parsing logic or in-memory representations. The latest transformer is a quantifier prenexer to test out different prenexing strategies, while the latest scripts generate a GraphViz tree of a formula or print unquantified variables. We are still adding more automated tests and new features, while evaluating what other areas could benefit from a common code-base. The project is available under MIT-license at:

https://gitlab.sai.jku.at/booleguru/booleguru

3 Literature

[1] "Limboole," Jan. 2023, [Online; accessed 26. Jan. 2023]. [Online]. Available: http://fmv.jku.at/limboole

[2] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437.

[3] C. Barrett, P. Fontaine, and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," 2016. [Online]. Available: https://smtlib.cs.uiowa.edu/

[1] https://www.lua.org/ and https://fennel-lang.org/