MiniCheck Documentation

Mario Roseneder, 0927370
Jan Müller, 12102339

June 14, 2023

Contents

1 Introduction

This document documents the MiniCheck implementation for the course *Advanced Functional Programming* by the TU Wien during the summer term of 2023.

The bounded LTL model checking algorithm was chosen as the extension to the core CTL model checker. This mode is accessible via the CLI as described in subsection 3.2.

Section 2 briefly summarizes the syntax for transition system (TS), as well as CTL and LTL formulas. Next, section 3 describes the development aspects of the project, including build tools, commands, and dependencies. The modules of the source code are described in section 4. Finally, section 5 covers the testing and coverage of the application.

2 Background

2.1 Syntax

All labels, i.e., atomic propositions, as well as state names can only consist of letters without numbers and whitespace. This does not limit their expressiveness, as a simple transformation of label names does not affect their boolean values. E.g., `nsoda === 0` is equivalent to `numberOfSodaIsZero`, as both propositions only represent a boolean value.

All formulas require brackets as described in the sections below. This approach was chosen to avoid ambiguity regarding operator precedence.

2.2 Transition Systems

The syntax of transition systems is displayed in the example below:

```
States:
-> pay
- soda
- select
- beer

Transitions:
- pay -> insert_coin -> select
- select ->    -> soda
- select ->    -> beer
- soda -> get_soda -> pay
```

– beer –> get_beer –> pay

The arrow indicates an initial state, while dashes indicate regular states. Each state is automatically labelled by itself, i.e., its name. Further labels, i.e., atomic propositions, can be specified by appending them to the state name, separated by a colon (e.g., `- beer: bier, bara`).

2.2.1 Validations

The following transition system validations are implemented:

- A transition system must have at least one initial state.
- The transitions may only specify defined states.
- All states must have an outgoing transition.

2.3 CTL Formulas

2.3.1 State Formulas

Formula	Syntax
true	`true`
$\neg\Phi$	`!(Φ)`
$\Phi_1 \wedge \Phi_2$	`(Φ1 && Φ2)`
$\Phi_1 \vee \Phi_2$	`(Φ1 || Φ2)`
$\Phi_1 \rightarrow \Phi_2$	`(Φ1 -> Φ2)`
$\Phi_1 \Longleftrightarrow \Phi_2$	`(Φ1 <-> Φ2)`
$\Phi_1 \oplus \Phi_2$	`(Φ1 xor Φ2)`
$\exists\Phi$	`(E Φ)`
$\forall\Phi$	`(A Φ)`

2.3.2 Path Formulas

Formula	Syntax
$\Phi_1 \mathcal{U} \Phi_2$	`(Φ1 U Φ2)`
$\circ\Phi$	`(X Φ)`
$\diamond\Phi$	`(F Φ)`
$\Box\Phi$	`(G Φ)`

2.3.3 Validations

The following CTL formula validations are implemented:

- All atomic propositions used in a formula must be defined in the transition system.

2.4 LTL Formulas

Formula	Syntax
true	`true`
$\neg\Phi$	`!(Φ)`
$\Phi_1 \wedge \Phi_2$	`(Φ1 && Φ2)`
$\Phi_1 \vee \Phi_2$	`(Φ1 || Φ2)`
$\Phi_1 \rightarrow \Phi_2$	`(Φ1 -> Φ2)`
$\Phi_1 \iff \Phi_2$	`(Φ1 <-> Φ2)`
$\Phi_1 \oplus \Phi_2$	`(Φ1 xor Φ2)`
$\Phi_1 \mathcal{U} \Phi_2$	`(Φ1 U Φ2)`
$\circ\Phi$	`(X Φ)`
$\diamond\Phi$	`(F Φ)`
$\Box\Phi$	`(G Φ)`

2.4.1 Validations

The following LTL formula validations are implemented:

- All atomic propositions used in a formula must be defined in the transition system.

3 Development

3.1 Build Tool

The program is implemented using the build tool Cabal (version 3.10.1.0) and GHC (version 9.2.5). During development, the Haskell Language Server (HSL) (version 1.9.1.0) was used.

3.1.1 Building

The executable can be built by invoking the command `cabal build`. However, installing it with the command `cabal install` is preferable as this approach enables the immediate execution of the program via the CLI.

3.2 Running

Running the program is possible by invoking `cabal run`, or `MiniCheck` if it was installed using `cabal install`. Appending `--help` prints the help text. The program has three modes, which are as follows:

minicheck validate TS_FILE Parse and validate the transition system found at `TS_FILE`.

minicheck ctl TS_FILE CTL_FORMULA Evaluate the CTL formula in the transition system found at `TS_FILE`.

minicheck ltl TS_FILE LTL_FORMULA BOUND Evaluate the LTL formula in the transition system found at `TS_FILE` with `BOUND` as the maximum path length.

3.3 Dependencies

Beyond the *base* and *containers* libraries, which are usually part of the Haskell prelude, this program uses *cmdargs* (version 0.10.22) and *parsec* (version 3.1.16.1). The former provides functionality for parsing and validating command line arguments, while the latter's monad parser forms the basis of the program's CTL, LTL, and TS parsers.

Further, the following language extensions are enabled:

- DerivedDataTypeable
- InstanceSigs
- NamedFieldPuns

4 Modules

The source code is distributed across the following three directories.

4.1 app

This directory contains the application logic that is invoked via the CLI. It is responsible for argument validation and calling the corresponding library methods.

4.2 lib

This directory contains the core logic of the program. The three sub-directories, CTL, LTL, and TS, each contain modules for their respective models, parsers, and validators. Further, CTL and LTL also each contain a module for their respective model checking algorithm. Finally, a Utils model contains common utilities required by multiple other modules.

4.3 test

This directory contains the spec files for the individual modules as well as an entry point for the test execution.

5 Testing

All parsers and model checking algorithms have been unit tested. The tests were used for test-driven development of the respective functionality. As such, they focus on validating the functionality of a single unit, e.g., a single formula construct.

The testing framework *hspec* is used to execute unit tests for the CTL, LTL, and TS parsers, as well as the model checker algorithms.

An expression coverage of 88% was achieved. The majority of untested code is the `Show` instance implementation of transition systems.