A Haskell Library for Automata Theory

Ari Zerner
ari@zerner.com

Prof. Matthew Fluet
mtf@cs.rit.edu

https://github.com/Ari-Zerner/hautomata

Automata Theory
Automata theory is the study of abstract machines called automata. The most commonly studied automata are finite state automata, pushdown automata, and Turing machines.

Haskell
Haskell is a purely functional programming language originally released in 1990. It has a powerful static type system with features such as algebraic data types, type classes, and parametric polymorphism. Haskell uses lazy evaluation by default.

Library Design
The library consists of four modules: Automaton, FSA, PDA, and TM.

The Automaton module defines type classes, data types, and utility functions that are used by the the different types of automata.

The FSA, PDA, and TM modules implement finite state automata, pushdown automata, and Turing machines, respectively. Each defines a deterministic version, a non-deterministic version, and a wrapper which may hold either a deterministic or non-deterministic automaton.

Every part of the library was designed with polymorphism in mind. Any types may be used for states and symbols, subject to minor constraints for the sake of efficiency.

Future Work
Several features could be implemented in order to improve the library:
- More convenience functions and custom operators, in order to simplify the syntax of calling the library
- Functions to convert between the different types of automata
- Operations such as equivalence, union, intersection, and concatenation (for appropriate types of automata)
- Validation on automata constructors
- Where appropriate, functions to generate, rather than accept, the languages defined by automata
- Other types of automata (e.g. cellular automata, tree automata)
- Automata wrappers such as a step counter
- Optimization of operations, especially step

Motivation
This library aims to become the foremost library for automata theory in Haskell, with a particular emphasis on pedagogical applications.

Design Trade-offs
Epsilon transitions are prohibited for non-deterministic pushdown automata, in order to guarantee that a step of a pushdown automaton runs in finite time.

The step function on Turing machines must be given a unit (empty tuple) as input, in order to maintain a consistent interface for step.

Because Turing machines are not always able to make a decision, two separate type classes were required: Decider and PartialDecider.

Verification
Implementations were verified using The Haskell Test Framework and QuickCheck. The ratio of source lines of code to test lines of code is around 1:1.

Haskell Test
Implementations were verified using The Haskell Test Framework and QuickCheck. The ratio of source lines of code to test lines of code is around 1:1.

Motivation
This library aims to become the foremost library for automata theory in Haskell, with a particular emphasis on pedagogical applications.

Design Trade-offs
Epsilon transitions are prohibited for non-deterministic pushdown automata, in order to guarantee that a step of a pushdown automaton runs in finite time.

The step function on Turing machines must be given a unit (empty tuple) as input, in order to maintain a consistent interface for step.

Because Turing machines are not always able to make a decision, two separate type classes were required: Decider and PartialDecider.

Verification
Implementations were verified using The Haskell Test Framework and QuickCheck. The ratio of source lines of code to test lines of code is around 1:1.