Tree Automata in RX (Extended Abstract)

Johannes Waldmann

Institut für Informatik, Universität Leipzig
Augustusplatz 10-11, D-04109 Leipzig, Germany
`http://www.informatik.uni-leipzig.de/~joe/`

Finite Tree Automata. A *finite tree automaton* [CDG+97,GS84] consists of a signature $\Sigma = \bigcup_k \Sigma_k$ (where Σ_k contains the function symbols of arity k), a finite state set Q, a set of accepting states $Q_F \subseteq Q$ and a transition relation $\Delta \subseteq \bigcup_k \left(\Sigma_k \times Q^k \times Q\right)$.

In a *top-down nondeterministic finite automaton* (`TNFA` in `RX`), the transition relation Δ is given by a partial function $\delta : Q \to \mathsf{Pow}(\bigcup_k \Sigma_k \times Q^k)$, where Pow stands for power set. If the transition relation Δ is given by a partial function $\delta : (\bigcup_k \Sigma_k \times Q^k) \to Q$ then we call the automaton *bottom-up deterministic* (`BDFA` in `RX`).

`RX` implements intersection, union, and complement of automata. `RX` computes the sets of reachable and productive states. `RX` transforms a `TNFA` into an equivalent `BDFA`, using the power set construction, and also minimizes the `BDFA` by computing equivalence classes of states.

Applications.. I have used `RX` in computations of rewrite closures of tree languages [Wal98].

Implementation of Finite Tree Automata. `RX` is written in the pure lazy functional programming language Haskell [Pe97].

The functions δ in `TNFA` and `BDFA` have finite domain, so they can be implemented by some lookup mechanism. `RX` uses the module `FiniteMap.hs` from the `ghc` library [Tea96], which internally employs 2-3-4-trees. It would be perfectly possible to use a different implementation since `FiniteMap` is exported as an abstract data type, so `RX` does not depend on implementation details.

`RX`'s automata types `TNFA` and `BDFA` are polymorphic in the state type. This leads to nice programs, for instance we can write down directly the cross product construction with type

```
cross :: (Ord a, Ord b) => TNFA a -> TNFA b -> TNFA (a, b)
```

or the power set construction. (The use of `FiniteMap` requires state types be instances of `Ord`.) For efficiency reasons, we normalize to type `TNFA Int` from time to time.

Active Literate Programming. `RX` input is a mix of text and program. When processing, text just gets copied, while `RX` expressions (programs) get formatted, according to rules specified in the input. Moreover, expressions can be marked *active*, so their value is computed and formatted as well. That way, literate `RX` can replace $\LaTeX\,2_\varepsilon$ math mode. Per default, inlined `RX` expressions are passive, while displayed `RX` expressions are active.

Bare (non-literate) RX input is given in a simple scripting language. It is parsed using combinators from the hbc library [Aug96], then the RX interpreter invokes the appropriate tree automata functions.

Problems with RX. The major problem is a fundamental one: we have to make automata deterministic from time to time, so the number of states explodes. The RX user could try to avoid determinization, or else I would have to find a more clever implementation of the power set construction. I fear this would have to break some abstraction boundaries.

Even if the RX user is patient enough to wait for a determinization to finish, the resulting BDFA usually is not human-readable. What I would like to output, in such cases, is a small (if not minimal) equivalent TNFA. But minimalization of nondeterministic automata is a hard theoretical problem (even for word automata) [ADN92], and good heuristics seem hard to find.

Modularization, Re-Use, and Efficiency. RX functions for input, output and automata operations can easily be called from user programs that are linked against the RX library. For example, I have written software for Set Constraints and Ground Tree Transducers that way.

Haskell's module system gives nice support here, for two reasons: when exporting objects, we can encapsulate details, and: modules don't have state (in fact nothing has state in a FP language), so we can call functions without worrying about unwanted interactions (side effects).

My approach in writing RX was to first make it correct, and then (possibly) optimize it. Of course there are parts of RX where the second step of that plan still waits to be done—but not necessarily by me: I trust that Haskell compilers (and their respective authors) are more clever than me when it comes to, say, inter module inlining and deforestation.

References

[ADN92] A. Arnold, A. Dicky, and M. Nivat. A note about minimal non-deterministic automata. *Bulletin of the EATCS*, 47:166–169, 1992.

[Aug96] Lennart Augustson. *The HBC compiler, Documentation for Release 0.9999.3.* Department of Computer Science, Chalmers University of Technology, Gteborg, ftp://ftp.cs.chalmers.se/pub/haskell/chalmers/, 1996.

[CDG+97] Hubert Comon, Max Dauchet, Rémi Gilleron, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications.* http://www.grappa.univ-lille3.fr/tata/, 1997.

[GS84] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[Pe97] John Peterson and Kevin Hammond (editors). *Report on the Programming Language Haskell, Version 1.4.* http://www.haskell.org/definition/, 1997.

[Tea96] The GHC Team. *The Glorious Glasgow Haskell Compilation System, Version 2.01, User's Guide.* Department Of Computer Science, University of Glasgow, ftp://ftp.dcs.gla.ac.uk/pub/haskell/glasgow/, 1996.

[Wal98] Johannes Waldmann. Normalization of S terms is decidable. In Tobias Nipkow, editor, *9th RTA*, number 1379 in LNCS, pages 138–150, 1998.