Conversion of Finite Tree Automata to Regular Tree Expressions by State Elimination

Tomáš Pecka, Jan Trávníček, and Jan Janoušek

Arbology Research Group
Dept. of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague

{tomas.pecka,jan.travnicek,jan.janousek}@fit.cvut.cz

August 31, 2020

Outline

Problem statement

- Let A be a nondeterministic bottom-up finite tree automaton (FTA).
- Transform A to an equivalent regular tree expression (RTE) E such that $L(E) = L(A)$.
  - Both FTA and RTE describe exactly the class of regular tree languages [CDG+07].
  - The problem is analogous to the (string) conversion from a finite automaton to a regular expression.

Related work

- Converting FTAs to RTEs using regular tree equations (Guellouma, Cherroun [GC18]).

Our approach

- Elimination of states (inspired by well known state elimination algorithm from strings [HMU03]).

Trees

- Trees are one of the fundamental data structures. Useful for hierarchical data (XML, AST, ...).
- Tree is defined by the means of graph theory.
- Our trees are rooted, ordered, labelled, and ranked.
- There is a hierarchy of tree languages, today we deal with *regular tree languages*.

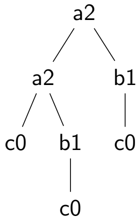


Figure: Tree t over ranked alphabet $\mathcal{A} = \{a2, b1, c0\}$. The number associated with the symbol is the arity of the symbol.

Finite Tree Automata I

- Standard computation model for regular tree languages.

Definition

Non-deterministic bottom-up (also frontier-to-root) finite tree automaton is a quadruple $A = (Q, \Sigma, \Delta, Q_F)$, where

- Q is the set of states,
- Σ is a ranked alphabet (symbols with non-negative arity),
- Δ is a set of transition rules (mapping $\Sigma_n \times Q^n \mapsto \mathcal{P}(Q)$), and
  - $a(q_1, \ldots, q_n) \to q$
- Q_F is the set of final states.

Finite Tree Automata II

- The computation moves from the leaves towards the root.
- Node a with arity n is assigned with state q if $a(q_1, \ldots, q_n) \to q \in \Delta$ and $q_1, \ldots, q_n$ are the states assigned to children of a.
- Tree is accepted by the automaton if the root is assigned with a final state.

Example

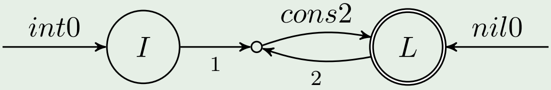


Figure: Example FTA accepting trees representing valid LISP lists consisting of int symbol.

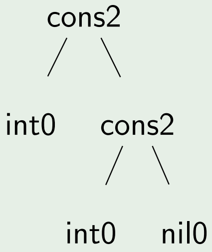


Figure: Run on an example tree accepted by the automaton.

Finite Tree Automata II

- The computation moves from the leaves towards the root.
- Node a with arity n is assigned with state q if $a(q_1, \ldots, q_n) \to q \in \Delta$ and $q_1, \ldots, q_n$ are the states assigned to children of a.
- Tree is accepted by the automaton if the root is assigned with a final state.

Example

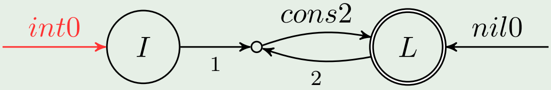


Figure: Example FTA accepting trees representing valid LISP lists consisting of int symbol.

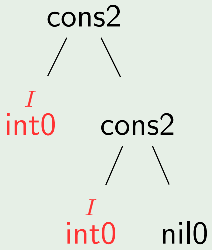


Figure: Run on an example tree accepted by the automaton.

Finite Tree Automata II

- The computation moves from the leaves towards the root.
- Node a with arity n is assigned with state q if $a(q_1, \ldots, q_n) \rightarrow q \in \Delta$ and $q_1, \ldots, q_n$ are the states assigned to children of a.
- Tree is accepted by the automaton if the root is assigned with a final state.

Example

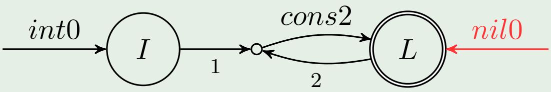


Figure: Example FTA accepting trees representing valid LISP lists consisting of int symbol.

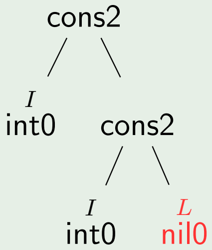


Figure: Run on an example tree accepted by the automaton.

Finite Tree Automata II

- The computation moves from the leaves towards the root.
- Node a with arity n is assigned with state q if $a(q_1, \ldots, q_n) \to q \in \Delta$ and $q_1, \ldots, q_n$ are the states assigned to children of a.
- Tree is accepted by the automaton if the root is assigned with a final state.

Example

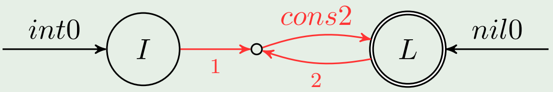


Figure: Example FTA accepting trees representing valid LISP lists consisting of int symbol.

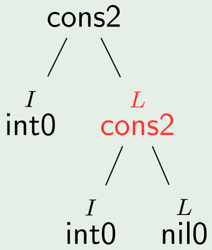


Figure: Run on an example tree accepted by the automaton.

Finite Tree Automata II

- The computation moves from the leaves towards the root.
- Node a with arity n is assigned with state q if $a(q_1, \ldots, q_n) \to q \in \Delta$ and $q_1, \ldots, q_n$ are the states assigned to children of a.
- Tree is accepted by the automaton if the root is assigned with a final state.

Example

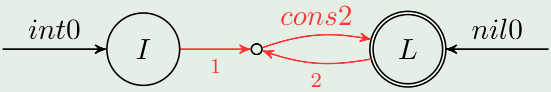


Figure: Example FTA accepting trees representing valid LISP lists consisting of int symbol.

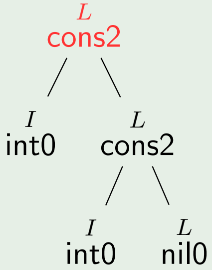


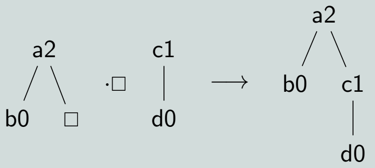
Figure: Run on an example tree accepted by the automaton.

Operations on Trees

Tree substitution

- Substituting occurrences of $\square_i$ by trees from L_i.
- Concatenating trees in specified places.
- $t\,\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}$

Example



Operations on Tree Languages

- Union: $L_1 + L_2 = L_1 \cup L_2$
- Concatenation: $L_1 \cdot_\square L_2 = \bigcup_{t \in L_1} \{t\,\{\square \leftarrow L_2\}\}$
- Closure: $L^{*,\square} = \bigcup_{n \geq 0} L^{n,\square}$.
  - $L^{0,\square} = \{\square\}, \quad L^{n+1,\square} = L \cdot_\square L^{n,\square}$

Regular Tree Expressions I

- Another way of describing regular tree languages.
- Analogous to regular (string) expressions.
- Defined as in TATA (Comon et al. [CDG$^+$07]): Alphabets of input symbols ($\mathcal{F}$) and substitution symbols ($\mathcal{K}$).

Definition

- $a \in \mathcal{F}_0$ is a RTE
- $\Box \in \mathcal{K}$ is a RTE
- If all E_i are RTEs, $a \in \mathcal{F}$, and $\Box \in \mathcal{K}$, then:
  - $a(E_1, \ldots, E_{arity(a)})$,
  - $E_1 + E_2$,
  - $E_1 \cdot_\Box E_2$, and
  - ${E_1}^{*,\Box}$ are RTEs.

Example

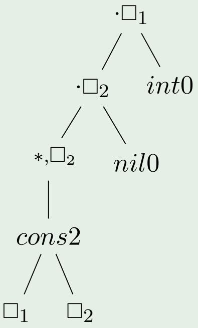
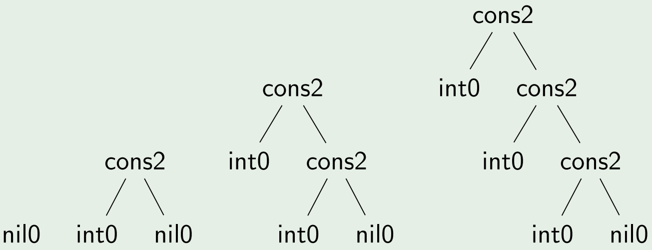


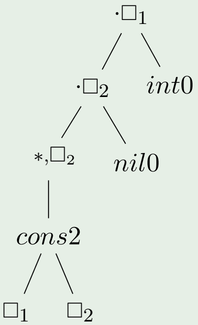
Figure: RTE for valid lists of integers in LISP.

$$\mathcal{F} = \{cons2, int0, nil0\}$$
$$\mathcal{K} = \{\Box_1, \Box_2\}$$



Regular Tree Expressions II

Example



$$\mathcal{F} = \{cons2, int0, nil0\}$$
$$\mathcal{K} = \{\square_1, \square_2\}$$

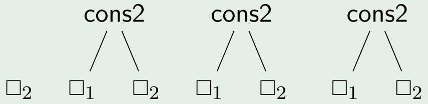
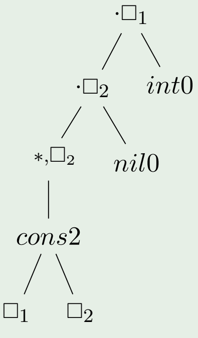


Figure: RTE for valid lists of integers in LISP.

Example



$$\mathcal{F} = \{cons2, int0, nil0\}$$
$$\mathcal{K} = \{\square_1, \square_2\}$$

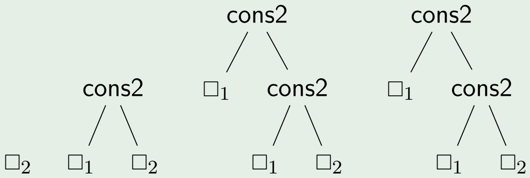
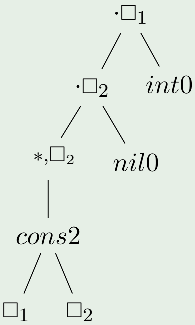


Figure: RTE for valid lists of integers in LISP.

Regular Tree Expressions II

Example



$$\mathcal{F} = \{cons2, int0, nil0\}$$
$$\mathcal{K} = \{\Box_1, \Box_2\}$$

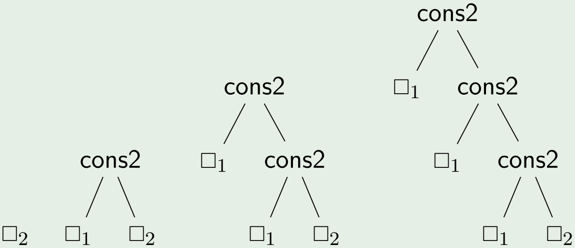


Figure: RTE for valid lists of integers in LISP.

Regular Tree Expressions II

Example

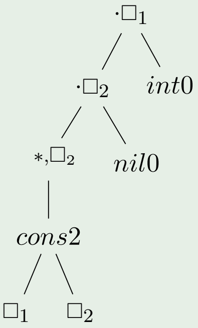
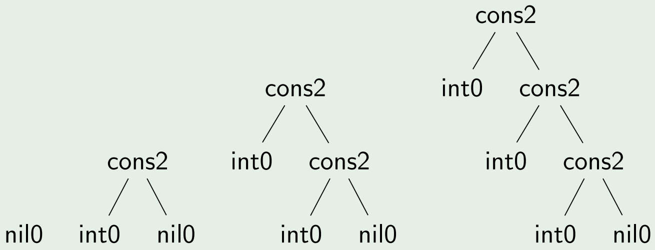


Figure: RTE for valid lists of integers in LISP.

$$\mathcal{F} = \{cons2, int0, nil0\}$$
$$\mathcal{K} = \{\Box_1, \Box_2\}$$



State elimination on FA

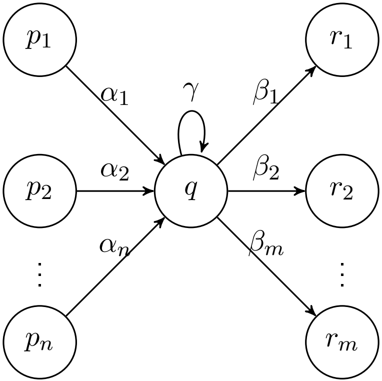


Figure: One step of a state elimination in FA.

- Generalised NFA: transitions use regular expressions instead of symbols
- Eliminate all states except the initial state and final state
  - Replace all paths through q with new transitions

    $$p_i \xrightarrow{\alpha_i \gamma^* \beta_j} r_j$$

  - Remove state q
- The path from the initial state to the final state is the equivalent regular expression.

State elimination on FTA

- Generalised FTA (GFTA): transitions use regular tree expressions instead of symbols.

From FTA to GFTA

- Create RTEs from symbols in the transitions mapping.
  - Source states of the transition will be children of the symbol.
  - Symbols corresponding to states are references to a language of a state.
  - Order of source states of the transition is no longer needed (now defined in the RTE).
- Transform the automaton to have only a one final state (useful later).

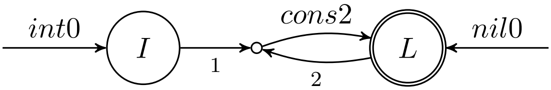


Figure: An example fragment of a FTA.

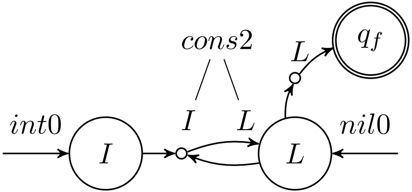


Figure: GFTA corresponding to the FTA on the left.

State elimination on FTA

Transition type

Classification of the GFTA's transitions with respect to a state q:

- incoming (q is a target, but not a source),
- outgoing (q is a source but not a target),
- looping (q is both a source and a target).

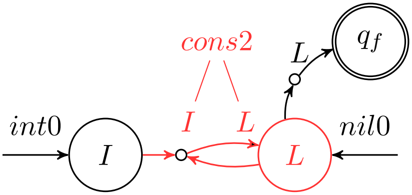


Figure: An example GFTA with a looping transition w.r.t. the state L.

Elimination of a single state from GFTA

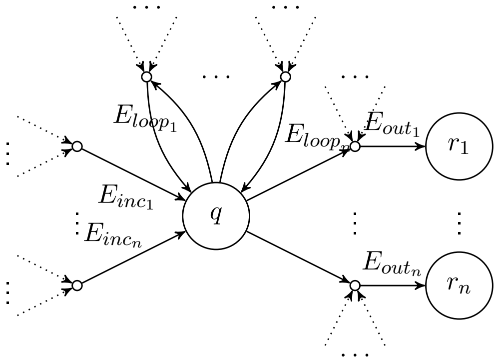
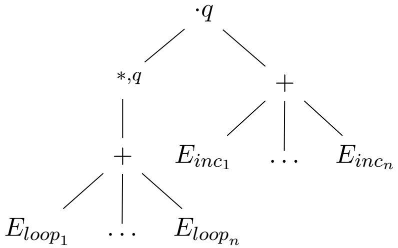


Figure: Situation when eliminating the state q of GFTA.

- E_{out} refers to state q (contains $q \in \mathcal{K}$).
- Language of state q can be seen as RTE with E_{loop} and E_{inc} "subRTEs" [CDG+07].



- Replace q in all E_{out} with this fragment ($O(1)$ if not replacing but using concatenation).

Elimination of a single state from GFTA

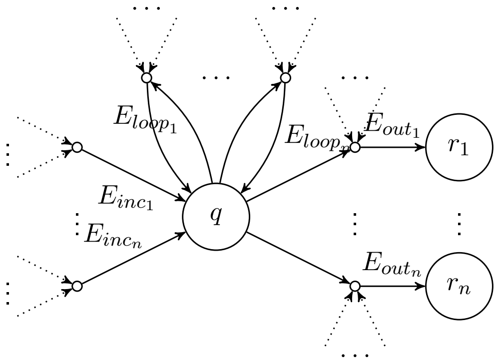


Figure: Situation when eliminating the state q of GFTA.

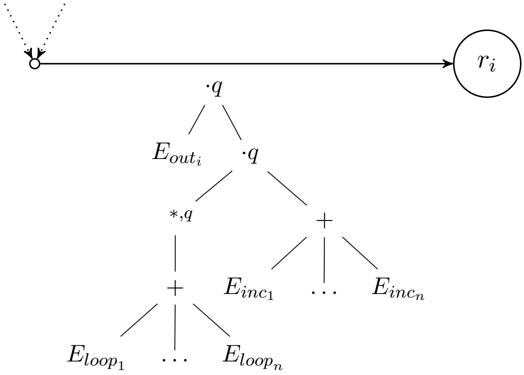


Figure: Modification of an outgoing edge.

Example

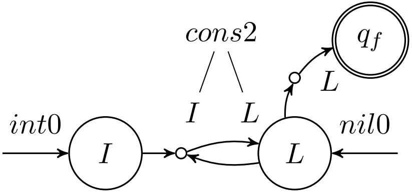


Figure: The original GFTA.

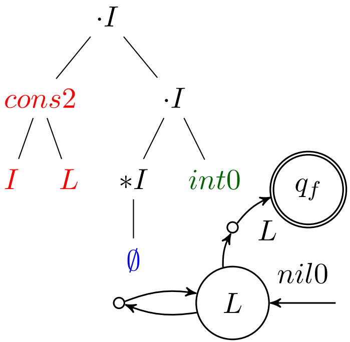


Figure: After eliminating state I.

Example

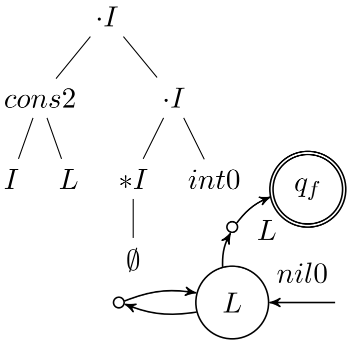


Figure: After eliminating state I.

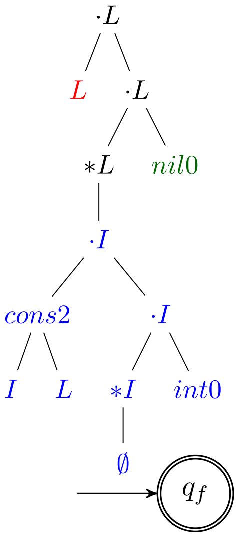


Figure: After eliminating state L.

State elimination on TA

Elimination Algorithm

1. Convert the input FTA to a GFTA.
2. Eliminate any non-final state using the approach on previous slide until a single-state GFTA remains.
3. The resulting RTE can be read from the transitions leading to the final state.

Time Complexity

For an input FTA $A = (Q, \Sigma, \Delta, Q_F)$ and corresponding GFTA $G = (Q \cup \{q_f\}, \Sigma, \Gamma, \{q_f\})$:

Conversion to GFTA is done in $O(|\Delta| + |Q_F|)$.

Eliminating a single state is $O(|Q| \cdot |\Gamma|)$, but both $|Q|$ and $|\Gamma|$ gradually decrease.

Total time consists of conversion and $|Q|$ invocations of elimination step: $O(|Q| \cdot |Q| \cdot (|\Delta| + |Q_F|))$.

Conclusion

- A simple algorithm for the construction of a regular tree expression(RTE) equivalent to given finite tree automaton (FTA) by eliminating states.
- Ideas come from a proof of RTE - FTA equivalence in [CDG$^+$07] and from the similar string algorithm [HMU03].
- Implementation in *Algorithms Library Toolkit* [ALT].

Future work:

- Is there a way of speeding up the elimination step?
- Good and bad elimination orders? Affects the size of the RTE.

5 Appendix: References

References I

- Algorithms Library Toolkit.
  `https://alt.fit.cvut.cz`.
- H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi.
  Tree automata techniques and applications, 2007.
  Release October 2007.
- Younes Guellouma and Hadda Cherroun.
  From tree automata to rational tree expressions.
  Int. J. Found. Comput. Sci., 29(6):1045–1062, 2018.
- John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman.
  Introduction to automata theory, languages, and computation (2. ed).
  Addison-Wesley, 2003.