Kleene Meets Church: Regular Expressions as Types

Bjørn Bugge Grathwohl, Ulrik Terp Rasmussen, Fritz Henglein

{bugge,dolle,henglein}@diku.dk http://www.diku.dk/kmc

Department of Computer Science
University of Copenhagen

Regular Expression Usage

Regular expressions are amongst the most widely used DSLs along with Excel and SQL. Despite this, they are ill-behaved from a programmer's perspective. The language interpretation of regular expressions as regular languages makes it difficult to use REs as a *data extraction* tool, even though this is what they are often used for.

For example, the expression

$a^\star b + (a+b)^\star$ is not the same as $(a+b)^\star$,

but they denote the same language

$$\mathcal{L}[\![a^\star b + (a+b)^\star]\!] = \mathcal{L}[\![(a+b)^\star]\!].$$

In Perl-style implementations, this problem is addressed with *subgroup matching*:

$$\underbrace{\texttt{(a*b)}}_{1} \;\texttt{|}\; \underbrace{\texttt{(a|b)*}}_{2}$$

but this behaves poorly in conjunction with Kleene stars:

$$\underbrace{\texttt{abababa}}_{\text{lost!}}\;\underbrace{\texttt{b}}_{2}\;\underbrace{?}_{1}$$

Type Interpretation

An RE E can be interpreted as a *type*, with $\mathcal{T}[\![E]\!]$ denoting a set of *parse trees*:

$$\begin{aligned}
\mathcal{T}[\![\mathbf{0}]\!] &= \emptyset \\
\mathcal{T}[\![\mathbf{1}]\!] &= \{()\} \\
\mathcal{T}[\![a]\!] &= \{a\} \\
\mathcal{T}[\![E_1E_2]\!] &= \mathcal{T}[\![E_1]\!] \times \mathcal{T}[\![E_2]\!] \\
\mathcal{T}[\![E_1+E_2]\!] &= \mathcal{T}[\![E_1]\!] + \mathcal{T}[\![E_2]\!] \\
\mathcal{T}[\![E_1^\star]\!] &= \{[v_1,\ldots,v_n] \mid v_i \in \mathcal{T}[\![E_1]\!]\}
\end{aligned}$$

A parser tells the programmer *how* the input string matches E; with $a^\star b + (a+b)^\star$ and "aaaaab" it is one of:

$$\mathsf{inl}\ \langle [a,a,a,a,a], b\rangle$$

$$\mathsf{inr}\ [\mathsf{inl}\ a, \mathsf{inl}\ a, \mathsf{inl}\ a, \mathsf{inl}\ a, \mathsf{inl}\ a, \mathsf{inr}\ b].$$

Define bit-codes $\mathcal{B}[\![\cdot]\!]$ for typed trees such that:

$$\forall E\ .\ \mathcal{T}[\![E]\!] \cong \mathcal{B}[\![E]\!] \subseteq \{0,1\}^\star$$

Automata Representation

Theorem 1 ([2, 3]). *Paths in a Thompson NFA correspond 1-to-1 to parse trees.*

Theorem 2 ([3]). *The greedy parse tree corresponds to the "leftmost" path.*

So: *bit-code of greedy parse* $\longleftrightarrow$ *leftmost path.* We can annotate a Thompson NFA to output bit-codes, producing an *aNFA*.

References

[1] Alur, R., Černý, P.: Expressiveness of Streaming String Transducers. *LIPIcs*, pp. 1–12. 2010.

[2] Brüggemann-Klein, A.: Regular Expressions into Finite Automata. *Theor. Comput. Sci.* 120(2), pp. 197–213. 1993.

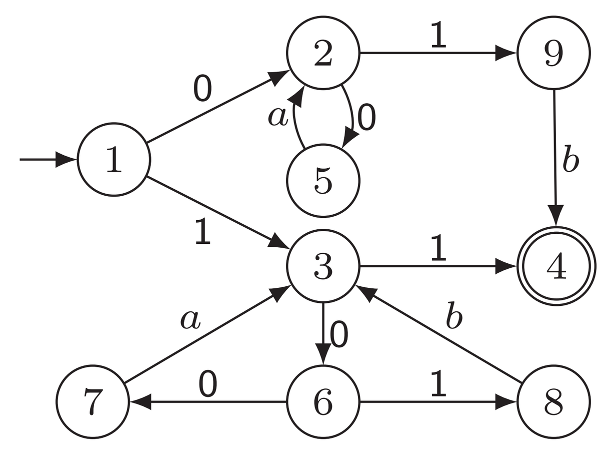
[3] Grathwohl, N.B.B., Henglein, F., Nielsen, L., Rasmussen, U.T.: Two-Pass Greedy Regular Expression Parsing. *Proc. CIAA 2013*, LNCS, vol. 7982, pp. 60–71. July 2013.

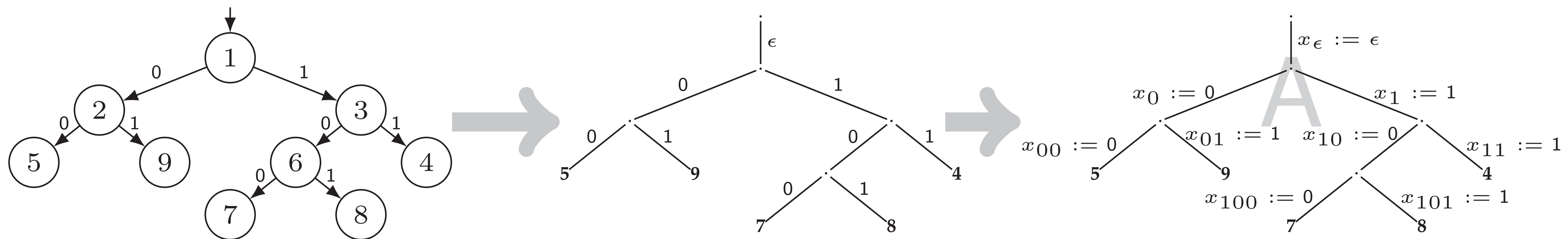
[4] Grathwohl, N.B.B., Henglein, F., Rasmussen, U.T.: Optimally Streaming Greedy Regular Expression Parsing. *Proc. ICTAC 2014*, LNCS, vol. 8687, pp. 224–240. September 2014.

[5] Henglein, F., Nielsen, L.: Regular Expression Containment: Coinductive Axiomatization and Computational Interpretation. *Proc. POPL 2011*, pp. 385–398. January 2011.

Path Trees

We maintain the set of paths through an aNFA in a *path tree*. Internal nodes represent states wherefrom paths differ. If we associate a buffer with each edge, it suffices to store the structure of the tree and its leaf nodes, not the internal nodes. As the internal nodes represent the points where paths differ, the contents of the root buffer prefixes all parse trees that can be produced by reading the remaining input. Hence, it can be output immediately.





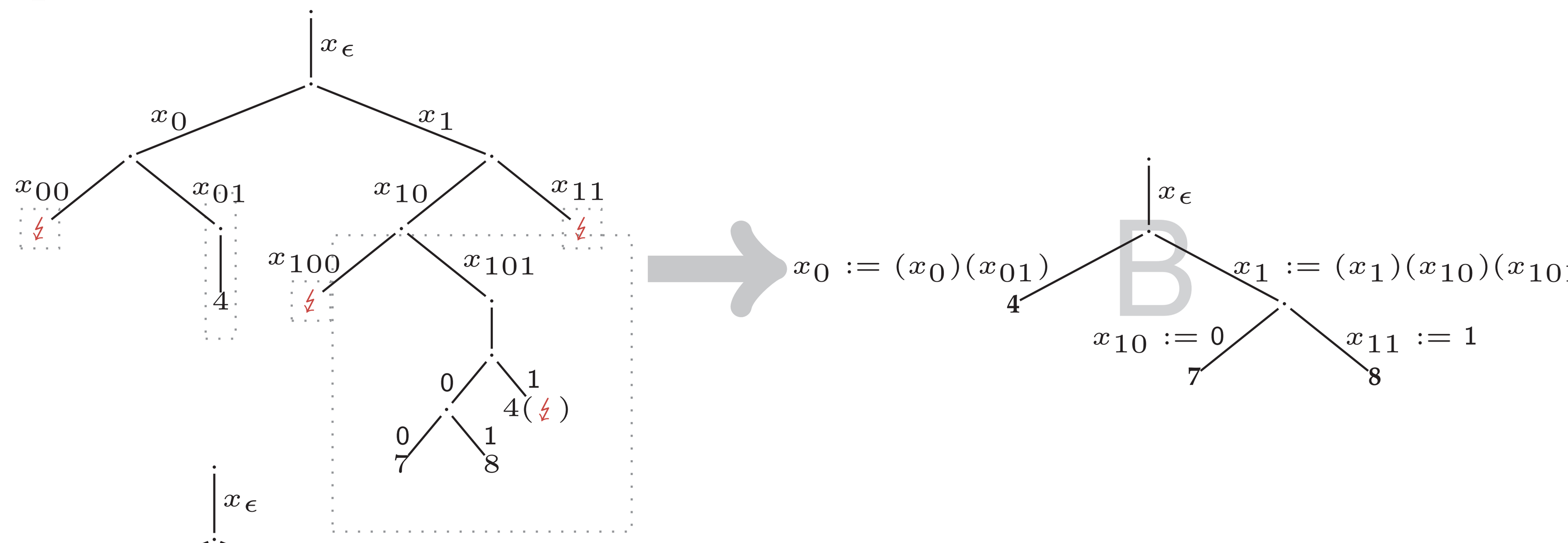
Abstract an ϵ-closure and leave only the labelled tree structure and the leaves.

Associate a buffer to each internal edge of the path tree.

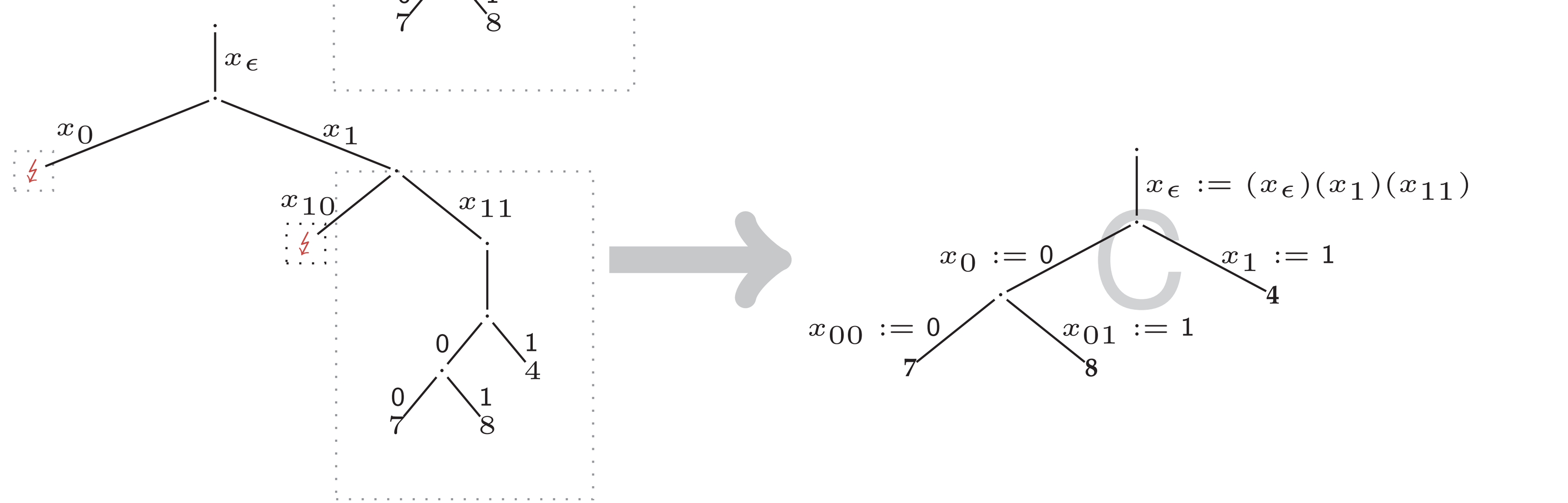
Simulation

For each input symbol, the path tree is expanded at the leaves by following the aNFA transitions. We maintain uniqueness of the leaves by marking repetitions as dead; this corresponds to the greedy-leftmost disambiguation strategy. After a tree has been expanded it is contracted by combining buffers on the paths from the root to the leaves.

After consuming one "b" some leaves are dead, marked ↯. The tree is then contracted.



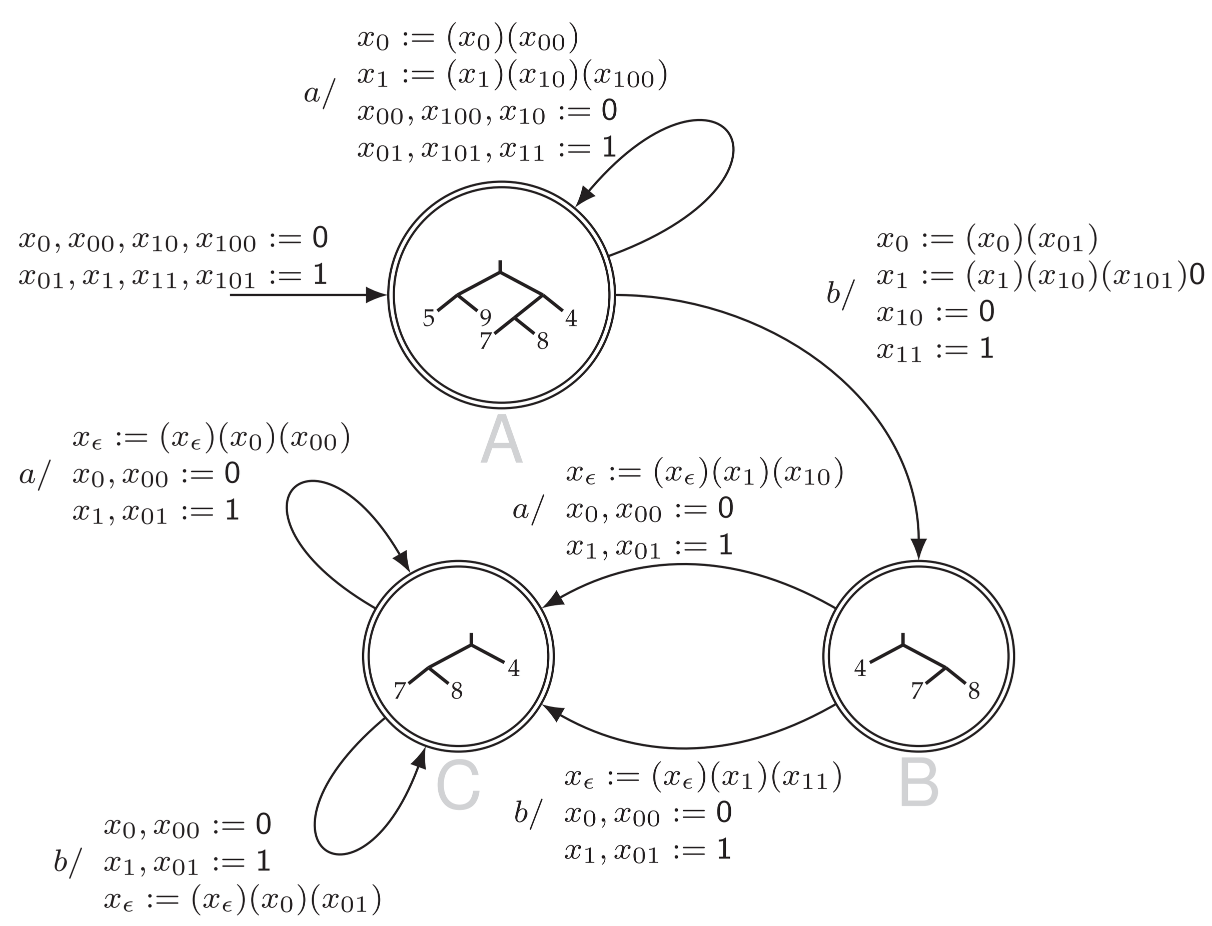
After consuming two "b"s and contraction.



Determinization

We can use the path tree simulation to determinize and disambiguate an aNFA, using the path trees as states and associating buffer updates with each transition. This corresponds to the streaming string transducers of Alur et al. [1]. The root buffer x_ϵ represents *output*. Such a transducer thereby does *streaming parsing* of regular expressions.

The pictured transducer outputs something immediately after the first "b" has been read and the next input symbol is known. Only then can we know which of $a^\star b$ and $(a+b)^\star$ parses the input.



Optimal Streaming

Let $C_E(w)$ be the set of strings prefixed by w that is in the language of E. The *optimally* streaming function can be formulated as follows:

Definition 1 (Optimally streaming function [4]). *The* optimally streaming *function corresponding to* $\mathsf{P}_E(\cdot)$ *is*

$$O_E(w) = \bigsqcap\{\mathsf{P}_E(w') \mid w' \in C_E(w)\}$$

Intuitively, at reading input w, an optimally streaming parsing function must output the *longest* prefix of the set of paths reaching an accepting state. It requires a PSPACE-complete analysis to get optimal streaming, but for non-pathological expressions our approach suffices.