Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the semantics of regular expression parsing in the wild ☆

Martin Berglund [a], Brink van der Merwe [b,*]

[a] Department of Computing Science, Umeå University, Sweden
[b] Department of Computer Science, Stellenbosch University, South Africa

ARTICLE INFO

Article history:
Received 30 November 2015
Received in revised form 7 September 2016
Accepted 11 September 2016
Available online xxxx

Keywords:
Regular expression matchers
Capturing groups
Prioritized transducers

ABSTRACT

We introduce prioritized transducers to formalize capturing groups in regular expression matching in a way that permits straightforward modeling of capturing in Java's [1] regular expression library. The broader questions of parsing semantics and performance are also considered. In addition, the complexity of deciding equivalence of regular expressions with capturing groups is investigated.

1. Introduction

Many regular expression matching libraries perform matching as a form of parsing by using *capturing groups,* and thus output what subexpression matched which substring [9]. This form of regular expression matching requires theoretical underpinnings different from classical regular expressions as defined in formal language theory. A popular implementation strategy used for performing regular expression matching (or parsing) with capturing groups, used for example in Java, .NET and the PCRE library [14], is a worst-case exponential time depth-first search strategy. A formal approach to matching with capturing groups can be obtained by using finite state transducers that output annotations on the input string to signify what subexpression matched which substring [16]. A complicating factor in this approach is introduced by the fact that the matching semantics dictates a single output string for each input string, obtained by using rules to determine a "highest priority" match among the potentially exponentially many possible ones (in contrast, [6] discusses non-deterministic capturing groups).

The pNFA (prioritized non-deterministic finite automaton) model of [3] (a similar formalism was also introduced later in [14]) provides the right level of abstraction to model the matching time behavior of regular expression matchers (at least in Java), as established experimentally in [18]. Also, for matchers based on an input directed depth first search, adding output to pNFA to obtain pTr (prioritized transducers) provides a way of modeling matching with capturing groups.

A regular expression to transducer (with regular lookahead) construction, for regular expressions using a Perl matching strategy, is presented in [16]. Our approach permits an analysis of matching semantics of a subset of the regular expressions supported in Java, but also makes it possible to model alternative matching semantics as found in matchers such as RE2 [7]. In Section 4, where we discuss how to convert regular expressions to pTr, it will become clear that converting regular expressions to pTr is a natural generalization of the Thomson construction for converting regular expressions to

☆ This article is a revised and extended version of [4].

* Corresponding author.
E-mail addresses: mbe@cs.umu.se (M. Berglund), abvdm@cs.sun.ac.za (B. van der Merwe).

[1] Java is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

http://dx.doi.org/10.1016/j.tcs.2016.09.006
0304-3975/

non-deterministic finite automata. We also discuss a linear-time matching algorithm for pTr (i.e. determining the image of input strings), where in contrast [10] presents a parsing algorithm operating directly on regular expressions.

The title of our paper was chosen intentionally to be very similar to the title of a blogpost by Russ Cox [7], in which he describes the functionality and performance of the regular expression matcher RE2. Cox states, without giving any theoretical arguments, that the regular expression matcher RE2 demonstrates that it is possible to use automata theory to implement almost all the features of a modern backtracking regular expression matcher. He further claims that because RE2 is rooted in the theoretical foundation of automata, it provides stronger guarantees on execution time. The aim of our paper is to provide an automata-based theoretical foundation for the basic functionality of modern regular expression matchers (with a focus on the Java regular expression standard library), from which it will follow that the acceptable execution times mentioned by Cox are indeed attainable.

Our main contribution is in defining matching of strings by regular expressions in such a way to incorporate what subexpression matched which substring and in showing that for a given regular expression, a pTr can be constructed that provides exactly the same matching information. Our other contributions are in providing complexity bounds for determining the output string produced by a pTr for a given input string and for equivalence checking of regular expressions when using our more general definition of regular expression matching.

The outline of the paper is as follows. In the next section we define prioritized automata and transducers. After this we discuss, with examples, the regular expression matching in Java, and also the POSIX standard. This motivates the approach we follow in adapting the standard Thompson construction for converting regular expressions to non-deterministic finite automata, from [17], to the more general setting of converting regular expressions to prioritized transducers. The following section gives a normal form for prioritized transducers, the so-called flattened prioritized transducers that simplifies discussions in the next section on deciding equivalence of and parsing with pTr.

2. Definitions

Let $\mathrm{dom}(f)$ and $\mathrm{range}(f)$ denote the domain and range of a function f, respectively. When unambiguous let a function f with $\mathrm{dom}(f) = S$ generalize to S^* and $\mathcal{P}(S)$ element-wise, where $\mathcal{P}(S)$ denotes the power set of the set S. The cardinality of a (finite) set S is denoted by $|S|$. We denote by $\mathbb{N}$ the set of natural numbers, i.e. the set $\{1, 2, 3, \ldots\}$. The empty string is denoted ε. An alphabet Σ is a finite set of symbols with $\varepsilon \notin \Sigma$. We denote $\Sigma \cup \{\varepsilon\}$ by Σ^ε. For any string w let $\pi_S(w)$ be the maximal subsequence of w containing only symbols from S (e.g. $\pi_{\{a,b\}}(abcdab) = abab$).

If $w_1 \in \Sigma_1^*$ and $w_2 \in \Sigma_2^*$, with Σ_1 and Σ_2 disjoint alphabets, then $w \in (\Sigma_1 \cup \Sigma_2)^*$ is in the *shuffle of* w_1 *and* w_2 if $\pi_{\Sigma_1}(w) = w_1$ and $\pi_{\Sigma_2}(w) = w_2$. The shuffle of two languages L_1 and L_2, over disjoint alphabets, is the shuffle of all pairs of words from L_1 and L_2 respectively.

For sequences $s = (z_{1,1}, \ldots, z_{1,n}) \ldots (z_{m,1}, \ldots, z_{m,n}) \in (Z_1 \times \ldots \times Z_n)^*$, we denote by $\sigma_i(s)$ the subsequence of tuples obtained from s by deleting duplicates of tuples in s and only keeping the first occurrence of each tuple, where equality of tuples is based only on the value of the ith component of a tuple (e.g. $\sigma_1((1,a)(2,a)(1,b)(3,b)(2,c)) = (1,a)(2,a)(3,b)$). For each $k > 1$, we denote by B_k the alphabet of k types of brackets, which is represented as $\{[_1,]_1, [_2,]_2, \ldots [_k,]_k\}$. The Dyck language D_k over the alphabet B_k is the set of strings representing well balanced sequences of brackets over B_k.

As usual, a regular expression over an alphabet Σ (where $\varepsilon \notin \Sigma$) is either an element of $\Sigma \cup \{\varepsilon, \emptyset\}$ or an expression of one of the forms $(E \mid E')$, $(E \cdot E')$, or (E^*), where E and E' are regular expressions. Some parentheses can be dropped with the rule that * (Kleene closure) takes precedence over $\cdot$ (concatenation), which takes precedence over $\mid$ (union). Further, outermost parentheses can be dropped, and $E \cdot E'$ can be written as EE'. The language of a regular expression E, denoted $\mathcal{L}(E)$, is obtained by evaluating E as usual, where $\emptyset$ stands for the empty language and $a \in \Sigma \cup \{\varepsilon\}$ for $\{a\}$. The *size* of E, denoted $|E|$, is the number of symbols appearing in E. A *capturing group* is any parenthesized subexpression, e.g. (E). Brackets in regular expressions are used both for precedence and capturing. The precise matching and capturing semantics follow from Section 4.

For later constructions we require a few different kinds of automata and transducers. First (non-)deterministic finite automata (and their runs when applied to strings), followed by the prioritized finite automata from [3], which are used to model the capturing behavior of (some of the most popular) regular expression matching libraries.

Definition 1. A *non-deterministic finite automaton* (NFA) is a tuple $A = (Q, \Sigma, q_0, \delta, F)$ where:

- Q is a finite set of *states;*
- Σ is the *input alphabet;*
- $q_0 \in Q$ is the *initial state;*
- $\delta : Q \times \Sigma^\varepsilon \to \mathcal{P}(Q)$ is the *transition function; and*
- $F \subseteq Q$ is the set of *final states.*

A is *ε-free* if $\delta(q, \varepsilon) = \emptyset$ for all q. A is *deterministic* if it is *ε-free* and $|\delta(q, \alpha)| \leq 1$ for all q and α. The *state size* of A is denoted by $|A|_Q$, and defined to be $|Q|$.

Please cite this article in press as: M. Berglund, B. van der Merwe, On the semantics of regular expression parsing in the wild, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.006