A SIMPLE UNIVERSAL PATTERN-MATCHING AUTOMATON

DANIEL J. BERNSTEIN

Abstract. Consider an infinite non-deterministic automaton with one state $\boxed{p}$ for each regular expression p; transitions $\boxed{q} \xrightarrow{c} \boxed{qS}$ whenever S is a character set containing c; and null transitions $\boxed{q} \Rightarrow \boxed{q\overline{r}}$, $\boxed{q\overline{r}r} \Rightarrow \boxed{q\overline{r}}$, $\boxed{qr} \Rightarrow \boxed{q(r' \cup r)}$, and $\boxed{qr'} \Rightarrow \boxed{q(r' \cup r)}$. If this automaton starts at the empty regular expression, then $\boxed{p}$ recognizes exactly the language defined by p, for every p. The subautomaton affecting $\boxed{p}$ has at most $1 + \operatorname{len} p$ states.

1. Introduction

This paper presents a nondeterministic infinite automaton that recognizes all regular expressions simultaneously. A portion of the automaton is shown in Figure 1 below.

The automaton has one state $\boxed{p}$ for each regular expression p, and no other states. The language recognized by $\boxed{p}$ is exactly the language defined by p. The subautomaton affecting $\boxed{p}$ has at most $1 + \operatorname{len} p$ states. Here $\operatorname{len} p$ is the number of non-parenthesis symbols in p; for example, the length of $\overline{?}\,\overline{\mathtt{xy}}\mathtt{xz}$ is 7, and the length of $((\mathtt{xy} \cup \mathtt{z})\mathtt{z}) \cup \mathtt{yyy}$ is 9.

Is it surprising that such an automaton exists? Of course not. It is well known that, for each p, there is a nondeterministic automaton recognizing p with at most $1 + \operatorname{len} p$ states. One can mechanically assign to each state a corresponding regular expression, and finally merge the automata for all p into a single infinite automaton that behaves as described above.

What *is* surprising about the automaton in this paper is that its definition is extremely short. There is one transition $\boxed{q} \xrightarrow{c} \boxed{qC}$ for each regular expression q, character c, and character set C containing c. There are null transitions $\boxed{q} \Rightarrow \boxed{q\overline{r}}$, $\boxed{q\overline{r}r} \Rightarrow \boxed{q\overline{r}}$, $\boxed{qr} \Rightarrow \boxed{q(r' \cup r)}$, and $\boxed{qr'} \Rightarrow \boxed{q(r' \cup r)}$, for all regular expressions q, r, r'. The automaton begins at $\boxed{()}$ where $()$ is the empty pattern. That's it.

These transitions are visibly correct, in the sense that any string recognized by $\boxed{p}$ is in the language defined by p. It is not as easy to prove that these transitions are adequate, in the sense that any string in the language defined by p is recognized by $\boxed{p}$. See Theorem 4.1 and Theorem 4.2. It also takes some work to prove that the subautomaton affecting p has at most $1 + \operatorname{len} p$ states. See Theorem 7.2. These proofs occupy the remaining sections of this paper.

Date: 20000806.

2000 *Mathematics Subject Classification.* Primary 68Q45.

Key words and phrases. Regular expressions, nondeterministic infinite automata.

The author was supported by the National Science Foundation under grant DMS–9600083.

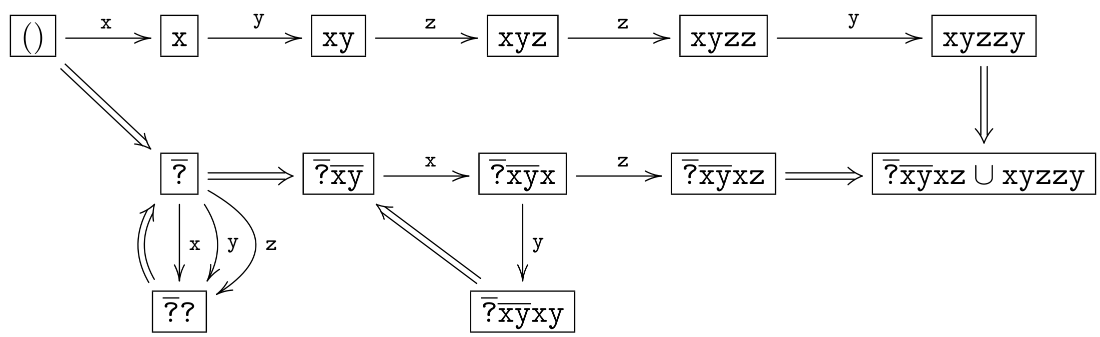

FIGURE 1. A portion of the automaton over the alphabet $\{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$.

Notation and terminology. The string $(c_1, \ldots, c_n)$ is abbreviated as "$c_1 \ldots c_n$". For example, "x", "xyz", and "xyzzy" are strings over the alphabet $\{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$.

A **constant** is a set of single-character strings. The set of all single-character strings is denoted ?. At the risk of confusion, I abbreviate the constant $\{$"c"$\}$ as c for each character c.

A **pattern algebra** is a set with an associative binary operation "composition" written $p, q \mapsto pq$, a neutral element for composition written $()$, a unary operation "closure" written $p \mapsto \overline{p}$, and a binary operation "union" written $p, q \mapsto p \cup q$. For example, the set of regular languages is a pattern algebra with composition $L, M \mapsto LM = \{st : s \in L, t \in M\}$; neutral element $() = \{$""$\}$; union equalling the usual set union; and closure $L \mapsto \overline{L} = L^0 \cup L^1 \cup \cdots$.

A **pattern** is an element of the free pattern algebra on the set of constants. In other words, a pattern is a formula built up from constants via union, closure, and composition, modulo redundant parentheses and the associativity of composition. Every pattern falls into one of the four forms $()$, qC, $q\overline{r}$, or $q(r' \cup r)$; here q, r', and r are patterns, and C is a constant.

I write $s \in p$, and say p **matches** s, to mean that s is contained in the language defined by p. Here p is a pattern and s is a string.

Historical notes. Thompson in [1] constructed an automaton recognizing p with $O(\operatorname{len} p)$ states. See [2] for a coherent survey of subsequent constructions. My construction is simpler than any of the constructions in [2].

I wrote down my automaton in June 1991, and distributed an implementation in a posting to `alt.sources` in January 1992. I was not familiar with the literature; the construction seemed so obvious that I assumed it was what everyone had always done. In April 1994, after reading a preliminary version of the taxonomy in [2], I announced my automaton in a posting to `comp.theory`.

2. PATTERN IMPLICATION

The relation "$p' \Rightarrow p$" is the transitive closure of the **basic implications** "$q \Rightarrow q\overline{r}$", "$q\overline{r}r \Rightarrow q\overline{r}$", "$qr \Rightarrow q(r' \cup r)$", and "$qr' \Rightarrow q(r' \cup r)$". In other words, $p' \Rightarrow p$ if and only if there is a chain $p' = p_0 \Rightarrow p_1 \Rightarrow \cdots \Rightarrow p_n = p$ of basic implications. In particular $p \Rightarrow p$.

For example, $() \Rightarrow \overline{?}$, and $\overline{?} \Rightarrow \overline{?}\,\overline{\mathtt{xy}}$, so $() \Rightarrow \overline{?}\,\overline{\mathtt{xy}}$.

Theorem 2.1. *If $p' \Rightarrow p$ and $s \in p'$ then $s \in p$.*

Proof. It suffices to check the basic implications: if $s \in q$ or $s \in q\overline{r}r$ then $s \in q\overline{r}$; and if $s \in qr$ or $s \in qr'$ then $s \in q(r' \cup r)$. □

Theorem 2.2. *If $p' \Rightarrow p$ then $p''p' \Rightarrow p''p$.*

Proof. This is a purely formal consequence of the definition: $p''q\overline{r}r \Rightarrow p''q\overline{r}$, $p''q \Rightarrow p''q\overline{r}$, $p''qr \Rightarrow p''q(r' \cup r)$, and $p''qr' \Rightarrow p''q(r' \cup r)$, so given a chain of basic implications we may prepend p'' to each term. □

3. The impl function

Define recursively

$$\operatorname{impl} p = \begin{cases} \{p\} & \text{if } p = () \text{ or } p = qC, \\ \operatorname{impl} qr' \cup \operatorname{impl} qr & \text{if } p = q(r' \cup r), \\ \operatorname{impl} q \cup \{q\overline{r}y : y \in \operatorname{impl} r, y \neq ()\} & \text{if } p = q\overline{r}. \end{cases}$$

For example, $\operatorname{impl} \overline{\texttt{?}\overline{\texttt{xy}}} = \left\{(), \overline{\texttt{??}}, \overline{\texttt{?}\overline{\texttt{xy}}}\texttt{xy}\right\}$.

Theorem 3.1. *If $x \in \operatorname{impl} p$ then $x \Rightarrow p$.*

Proof. Induct on $\operatorname{len} p$. First, if $p = ()$ or $p = qC$ then $x = p$ so $x \Rightarrow p$.

Second, say $p = q(r' \cup r)$. If $x \in \operatorname{impl} qr'$ then by induction $x \Rightarrow qr' \Rightarrow p$. If $x \in \operatorname{impl} qr$ then by induction $x \Rightarrow qr \Rightarrow p$.

Third, say $p = q\overline{r}$. If $x \in \operatorname{impl} q$ then by induction $x \Rightarrow q \Rightarrow p$. If $x = q\overline{r}y$, with $y \in \operatorname{impl} r$, then by induction $y \Rightarrow r$, so $x = q\overline{r}y \Rightarrow q\overline{r}r \Rightarrow p$ by Theorem 2.2. □

Theorem 3.2. *If $x \in \operatorname{impl} p$ then x is empty or ends with a constant.*

Proof. Induct on $\operatorname{len} p$. First, if $p = ()$ or $p = qC$ then $x = p$.

Second, if $p = q(r' \cup r)$ then $x \in \operatorname{impl} qr'$ or $x \in \operatorname{impl} qr$; by induction x is empty or ends with a constant.

Third, say $p = q\overline{r}$. If $x \in \operatorname{impl} q$ then by induction x is empty or ends with a constant. If $x = q\overline{r}y$, with $() \neq y \in \operatorname{impl} r$, then by induction y ends with a constant, so x does too. □

Theorem 3.3. *If $s \in p$ then $s \in x$ for some $x \in \operatorname{impl} p$.*

Proof. Induct on $\operatorname{len} p$. First, if $p = ()$ or $p = qC$ then $p \in \operatorname{impl} p$ so there is nothing to prove.

Second, if $p = q(r' \cup r)$ then $s \in qr'$ or $s \in qr$. So by induction $s \in x$ where $x \in \operatorname{impl} qr'$ or $x \in \operatorname{impl} qr$. Either way $x \in \operatorname{impl} p$.

Third, if $p = q\overline{r}$ then $s = t't$ with $t' \in q$, $t \in \overline{r}$. If $t =$ “” then $s = t' \in q$ so by induction there is an $x \in \operatorname{impl} q$ with $s \in x$; and $x \in \operatorname{impl} p$. If $t \neq$ “” we may write $t = uv$ where $u \in \overline{r}$ and $v \in r$, $v \neq$ “”. By induction there is a $y \in \operatorname{impl} r$ with $v \in y$; since v is nonempty we cannot have $y = ()$. Thus $q\overline{r}y \in \operatorname{impl} p$, and $s = t'uv \in q\overline{r}y$ as desired. □

4. Consequences of impl

Theorem 4.1 says that my automaton works for the empty string. Theorem 4.2 says that, if my automaton works for a string t, then it also works for t plus any character.

Theorem 4.1. “” $\in p$ *if and only if* $() \Rightarrow p$.

Proof. Say $() \Rightarrow p$. Certainly "" $\in ()$; by Theorem 2.1, "" $\in p$.

Conversely, say "" $\in p$. By Theorem 3.3, "" $\in x$ for some $x \in \operatorname{impl} p$. Since "" $\in x$, x cannot end with a constant; thus, by Theorem 3.2, $x = ()$. By Theorem 3.1, $() = x \Rightarrow p$. □

Theorem 4.2. *t"c" $\in p$ if and only if there exist q and C such that $t \in q$, "c" $\in C$, and $qC \Rightarrow p$.*

Proof. If $t \in q$ and "c" $\in C$ then t"c" $\in qC$; if also $qC \Rightarrow p$ then t"c" $\in p$ by Theorem 2.1.

Conversely, say t"c" $\in p$. By Theorem 3.3, t"c" $\in x$ for some $x \in \operatorname{impl} p$. Since t"c" $\neq$ "", x cannot be empty; thus, by Theorem 3.2, x ends with a constant. Write $x = qC$. Then $t \in q$ and "c" $\in C$. Finally $qC = x \Rightarrow p$ by Theorem 3.1. □

5. Left subpatterns

The relation "p' is a left subpattern of p" is the transitive closure of the relations "q is a left subpattern of qC", "$q\overline{r}r$ is a left subpattern of $q\overline{r}$", "q is a left subpattern of $q\overline{r}$", "qr is a left subpattern of $q(r' \cup r)$", and "qr' is a left subpattern of $q(r' \cup r)$".

In other words, p' is a left subpattern of p if and only if $\boxed{p'}$ affects $\boxed{p}$ in my automaton. Therefore, if $s \in p$, then any prefix s' of s satisfies $s' \in p'$ for some left subpattern p' of p.

Example: $\overline{\texttt{?}}\texttt{?}$ is a left subpattern of $\overline{\texttt{?}}$, which is a left subpattern of $\overline{\texttt{?}}\overline{\texttt{xy}}$, which is a left subpattern of $\overline{\texttt{?}}\overline{\texttt{xy}}\texttt{x}$, which is a left subpattern of $\overline{\texttt{?}}\overline{\texttt{xy}}\texttt{xz}$. So $\overline{\texttt{?}}\texttt{?}$ is a left subpattern of $\overline{\texttt{?}}\overline{\texttt{xy}}\texttt{xz}$.

6. The left function

Define $\operatorname{left}_0 p = p$. Define $\operatorname{left}_{n+1} p$ for each n as follows:

$$\operatorname{left}_{n+1} p = \begin{cases} \text{undefined} & \text{if } p = () \\ \operatorname{left}_n q & \text{if } p = qC \\ \operatorname{left}_n q\overline{r}r & \text{if } p = q\overline{r}, n < \operatorname{len} r \\ \operatorname{left}_{n-\operatorname{len} r} q & \text{if } p = q\overline{r}, n \geq \operatorname{len} r \\ \operatorname{left}_n qr & \text{if } p = q(r' \cup r), n < \operatorname{len} r \\ \operatorname{left}_{n-\operatorname{len} r} qr' & \text{if } p = q(r' \cup r), n \geq \operatorname{len} r. \end{cases}$$

For example, the values of $\operatorname{left}_n \overline{\texttt{?}}\overline{\texttt{xy}}\texttt{xz}$, as n increases from 0 through 7, are $\overline{\texttt{?}}\overline{\texttt{xy}}\texttt{xz}$, $\overline{\texttt{?}}\overline{\texttt{xy}}\texttt{x}$, $\overline{\texttt{?}}\overline{\texttt{xy}}$, $\overline{\texttt{?}}\overline{\texttt{xy}}\texttt{xy}$, $\overline{\texttt{?}}\overline{\texttt{xy}}\texttt{x}$, $\overline{\texttt{?}}$, $\overline{\texttt{?}}\texttt{?}$, and $()$. For $n > 7$, $\operatorname{left}_n \overline{\texttt{?}}\overline{\texttt{xy}}\texttt{xz}$ is undefined.

The reader may enjoy verifying certain facts not needed in this paper: $\operatorname{left}_{\operatorname{len} p} p = ()$; if $n < \operatorname{len} p$ then $\operatorname{left}_n p$ is defined and nonempty; if $n \leq \operatorname{len} p$ then $\operatorname{left}_n p'p = p' \operatorname{left}_n p$; if $\operatorname{left}_n p$ is defined then it is a left subpattern of p.

Theorem 6.1. *If $n > \operatorname{len} p$ then $\operatorname{left}_n p$ is undefined.*

Proof. Induct on n. Note that $n > 0$.

If $p = ()$ then $\operatorname{left}_n p$ is undefined. If $p = qC$ then $n > n - 1 > \operatorname{len} q$ so $\operatorname{left}_{n-1} q$ is undefined by induction; so $\operatorname{left}_n p = \operatorname{left}_{n-1} q$ is undefined. If $p = q\overline{r}$ then $n > n - 1 - \operatorname{len} r > \operatorname{len} q$ so $\operatorname{left}_{n-1-\operatorname{len} r} q$ is undefined by induction; so $\operatorname{left}_n p = \operatorname{left}_{n-1-\operatorname{len} r} q$ is undefined. If $p = q(r' \cup r)$ then $n > n - 1 - \operatorname{len} r > \operatorname{len} qr'$ so $\operatorname{left}_{n-1-\operatorname{len} r} qr'$ is undefined by induction; so $\operatorname{left}_n p = \operatorname{left}_{n-1-\operatorname{len} r} qr'$ is undefined. □

Theorem 6.2. $\operatorname{left}_n p'p = \operatorname{left}_{n-\operatorname{len} p} p'$ *if* $n \geq \operatorname{len} p$.

Proof. Induct on n.

If $p = ()$ then $\operatorname{len} p = 0$ and $p'p = p'$ as desired. If $p = qC$ then $n > n-1 \geq \operatorname{len} q$ so

$$\operatorname{left}_n p'p = \operatorname{left}_{n-1} p'q = \operatorname{left}_{n-1-\operatorname{len} q} p' = \operatorname{left}_{n-\operatorname{len} p} p'$$

by induction. If $p = q\overline{r}$ then $n > n - 1 - \operatorname{len} r \geq \operatorname{len} q$ so

$$\operatorname{left}_n p'p = \operatorname{left}_{n-1-\operatorname{len} r} p'q = \operatorname{left}_{n-1-\operatorname{len} r-\operatorname{len} q} p' = \operatorname{left}_{n-\operatorname{len} p} p'$$

by induction. If $p = q(r' \cup r)$ then $n > n - 1 - \operatorname{len} r \geq \operatorname{len} qr'$ so

$$\operatorname{left}_n p'p = \operatorname{left}_{n-1-\operatorname{len} r} p'qr' = \operatorname{left}_{n-1-\operatorname{len} r-\operatorname{len} qr'} p' = \operatorname{left}_{n-\operatorname{len} p} p'$$

by induction. □

7. The J function

Theorem 7.2 states that all left subpatterns of p are values of $\operatorname{left}_n p$ for $n \in \{0, 1, \ldots, \operatorname{len} p\}$. Thus the number of states affecting $\boxed{p}$ in my automaton is at most $1 + \operatorname{len} p$.

Define $J(p, m, 0) = m$. For each n define $J(p, m, n+1) =$

$$\begin{cases} \text{undefined} & \text{if } p = () \\ J(q, m, n) + 1 & \text{if } p = qC \\ J(q\overline{r}r, m, n) + 1 & \text{if } p = q\overline{r},\ n < \operatorname{len} r,\ J(q\overline{r}r, m, n) < \operatorname{len} r \\ J(q\overline{r}r, m, n) - \operatorname{len} r & \text{if } p = q\overline{r},\ n < \operatorname{len} r,\ J(q\overline{r}r, m, n) \geq \operatorname{len} r \\ J(q, m, n - \operatorname{len} r) + \operatorname{len} r + 1 & \text{if } p = q\overline{r},\ n \geq \operatorname{len} r \\ J(qr, m, n) + 1 & \text{if } p = q(r' \cup r),\ n < \operatorname{len} r,\ J(qr, m, n) < \operatorname{len} r \\ J(qr, m, n) + \operatorname{len} r' + 1 & \text{if } p = q(r' \cup r),\ n < \operatorname{len} r,\ J(qr, m, n) \geq \operatorname{len} r \\ J(qr', m, n - \operatorname{len} r) + \operatorname{len} r + 1 & \text{if } p = q(r' \cup r),\ n \geq \operatorname{len} r. \end{cases}$$

Theorem 7.1. $\operatorname{left}_m \operatorname{left}_n p = \operatorname{left}_{J(p,m,n)} p$ *if* $\operatorname{left}_n p$ *is defined.*

Proof. If $n = 0$ then $J(p, m, n) = m$ and $\operatorname{left}_n p = p$; thus $\operatorname{left}_{J(p,m,n)} p = \operatorname{left}_m p = \operatorname{left}_m \operatorname{left}_n p$.

Now induct on n. Assume $\operatorname{left}_{n+1} p$ is defined. Note that $n + 1 \leq \operatorname{len} p$ by Theorem 6.1, so $p \neq ()$. I will show that $\operatorname{left}_m \operatorname{left}_{n+1} p = \operatorname{left}_{J(p,m,n+1)} p$.

1. Say $p = qC$. Write $j = J(q, m, n)$. Then

$$\operatorname{left}_m \operatorname{left}_{n+1} p = \operatorname{left}_m \operatorname{left}_n q = \operatorname{left}_j q = \operatorname{left}_{j+1} p.$$

2. Say $p = q\overline{r}$ and $n < \operatorname{len} r$. Write $j = J(q\overline{r}r, m, n)$. Observe that

$$\operatorname{left}_m \operatorname{left}_{n+1} p = \operatorname{left}_m \operatorname{left}_n q\overline{r}r = \operatorname{left}_j q\overline{r}r.$$

If $j < \operatorname{len} r$ then $\operatorname{left}_j q\overline{r}r = \operatorname{left}_{j+1} p$; else $\operatorname{left}_j q\overline{r}r = \operatorname{left}_{j-\operatorname{len} r} q\overline{r} = \operatorname{left}_{j-\operatorname{len} r} p$.

3. Say $p = q\overline{r}$ and $n \geq \operatorname{len} r$. Write $j = J(q, m, n - \operatorname{len} r)$. Then

$$\operatorname{left}_m \operatorname{left}_{n+1} p = \operatorname{left}_m \operatorname{left}_{n-\operatorname{len} r} q = \operatorname{left}_j q = \operatorname{left}_{j+\operatorname{len} r+1} p.$$

4. Say $p = q(r' \cup r)$ and $n < \operatorname{len} r$. Write $j = J(qr, m, n)$. Observe that

$$\operatorname{left}_m \operatorname{left}_{n+1} p = \operatorname{left}_m \operatorname{left}_n qr = \operatorname{left}_j qr.$$

If $j < \operatorname{len} r$ then $\operatorname{left}_j qr = \operatorname{left}_{j+1} p$; otherwise

$$\operatorname{left}_j qr = \operatorname{left}_{j-\operatorname{len} r} q = \operatorname{left}_{j-\operatorname{len} r+\operatorname{len} r'} qr' = \operatorname{left}_{j+\operatorname{len} r'+1} p.$$

5. Say $p = q(r' \cup r)$ and $n \geq \operatorname{len} r$. Write $j = J(qr', m, n - \operatorname{len} r)$. Then

$$\operatorname{left}_m \operatorname{left}_{n+1} p = \operatorname{left}_m \operatorname{left}_{n-\operatorname{len} r} qr' = \operatorname{left}_j qr' = \operatorname{left}_{j+\operatorname{len} r+1} p.$$

□

Theorem 7.2. *If p' is a left subpattern of p then $p' = \operatorname{left}_n p$ for some n.*

Proof. If $p = qC$ then $q = \operatorname{left}_1 p$.

If $p = q\bar{r}$ then $q = \operatorname{left}_{\operatorname{len} r+1} p$. Furthermore $q\bar{r}r = \operatorname{left}_1 p$ if $r \neq ()$; $q\bar{r}r = \operatorname{left}_0 p$ if $r = ()$.

If $p = q(r' \cup r)$ then $qr' = \operatorname{left}_{\operatorname{len} r+1} p$. Furthermore $qr = \operatorname{left}_1 p$ if $r \neq ()$. If $r = ()$ then $qr = q = \operatorname{left}_{\operatorname{len} r'} qr' = \operatorname{left}_{\operatorname{len} r'+1} p$ by Theorem 6.2.

Finally, by Theorem 7.1, the relation "p' is $\operatorname{left}_n p$ for some n" is transitive. □

References

[1] Ken Thompson, *Regular expression search algorithm*, Communications of the ACM **11** (1968), 419–422.

[2] Bruce W. Watson, *Taxonomies and toolkits of regular language algorithms*, Ph.D. thesis, Eindhoven University of Technology, 1995.

Department of Mathematics, Statistics, and Computer Science, The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: djb@pobox.com