Transductive regular expression

Intro

The idea behind **trre** is very similar to regular expressions (**re**). But where **re** defines a regular language the **trre** defines a binary relation between two regular languages. To check if a string corresponds to a given regular expression normally we construct an abstract machine we call finite state automaton or finite state acceptor (FSA). There is an established formal correspondence between class of **re** and the class of FSA. For each expression in **re** we can find automaton. And for each automaton we can fine an expression in **re**.

For **trre** the class of FSA is not enough. Instead we use finite state transducer (FST) as underlying matching engine. It is similar to the acceptor but for each transition we have two labels. One defines an input character and other defines output. The current article establish correspondence between **trre** and FST similar to **re** and FSA.

To do this we need to define the trre language formaly and then prove that for every **trre** expression there is a corresponding FST and vise versa. But first, let's define the **re** language and then extend it to the **trre**.

RE language specification

Definition. Let $\mathcal{A}$ be an finite set augmented with symbols $\emptyset$ and ε. We call it an alphabet. Then **RE** language will be set of words over the alphabet $\mathcal{A}$ and symbols $+, \cdot, *$ defined inductively:

1. symbols $\varepsilon, \emptyset,$ and $a \in \mathcal{A}$ are regular expressions
2. for any regular expressions r and s the words $r + s$, $r \cdot s$ and $r*$ are regular expressions.

Definition. For a given regular expression r the language of r denoted as $L(r)$ is a set (may be infinite) of finite words defined inductively:

1. for trre expressions $\varepsilon, \emptyset, a \in \mathcal{A}$
   - $L(\varepsilon) = \{\varepsilon\}$
   - $L(\emptyset) = \emptyset$
   - $L(a) = \{a\}$
2. for regular expressions r, s
   - $L(r + s) = L(r) \cup L(s)$
   - $L(r \cdot s) = L(r) \cdot L(s) = \{a \cdot b \mid a \in L(r), b \in L(s)\}$
   - $L(r*) = L(r)* = \sum_{n \geq 0} r^n = \varepsilon + r + r \cdot r + r \cdot r \cdot r + ...$

The operation $r*$ we call Kleene star or Kleene closure.

Examples.
- $(a + b)c$
- $a * b *$

TRRE language specification

Definition. Let R be a set of regular expressions over alphabet $\mathcal{A}$ and S be a set of regular expression over alphabet $\mathcal{B}$. We refer to $\mathcal{A}$ as input alphabet and $\mathcal{B}$ as output alphabet. Transductive regular expressions `trre` will be a pair of regular expression combined with a delimeter symbol ':':
1. for regular expressions $r \in R, s \in S$, the string $r : s$ is a transductive regular expression
2. for transductive regular expressions t, u expressions $t \cdot u$, $t + u$, $t*$ are transductive regular expressions

Definition. For a given transductive regular expression t the language of t denoted as $L(t)$ is a set of **pairs** of words defined inductively:

1. for a transductive regular expression $r : s$ where r, s are the regular expressions the language $L(r : s) = L(r) \times L(s) = \{(a, b) \mid a \in L(r), b \in L(s)\}$, i.e. the direct product of languages $L(r)$ and $L(s)$.
2. for transudctive regular expression t, u:
   - $L(t \cdot u) = L(t) \cdot L(u) = \{(a \cdot b, x \cdot y) \mid (a, x) \in L(t), (b, y) \in L(u)\}$
   - $L(t + u) = L(t) \cup L(u)$
   - $L(t^*) = L(t)^* = \sum_{n \geq 0} t^n$.

Some properties of TRRE language

1. $a : b = a : \varepsilon \cdot \varepsilon : b = \varepsilon : b \cdot a : \varepsilon$

Proof:

$$\begin{aligned}
L(a : b) &= \{(u, v) \mid u \in L(a), v \in L(b)\} && \text{by definition of } a : b \text{ language} \\
&= \{(u \cdot \varepsilon), (\varepsilon \cdot v) \mid u \in L(a), v \in L(b)\} \\
&= \{(u \cdot \varepsilon), (\varepsilon \cdot v) \mid (u, \varepsilon) \in L(a : \varepsilon), v \in L(\varepsilon : b)\} \\
&= L(a : \varepsilon) \cdot L(\varepsilon : b) && \text{by definition of concatenation} \\
&= L(a : \varepsilon \cdot \varepsilon : b)
\end{aligned}$$

2. $(a + b) : c = a \cdot c + b \cdot c$

Proof:

$$\begin{aligned}
L((a + b) : c) &= L(a + b) \times L(c) && \text{by definition of } a : b \text{ language} \\
&= (L(a) \cup L(b)) \times L(c) && \text{by definition of } a + b \text{ language} \\
&= L(a) \times L(c) \cup L(b) \times L(c) && \text{cartesian product of sets is distributive over union} \\
&= L(a : c) \cup L(b : c) \\
&= L(a : c + b : c)
\end{aligned}$$

3. $a : (b + c) = a \cdot b + a \cdot c$

Proof: same as in 2

4. $a^* : \varepsilon = (a : \varepsilon)^*$

Proof:

$$\begin{aligned}
a^* : \varepsilon &= \left(\sum_{n \geq 0} a^n\right) : \varepsilon && \text{by definition of *} \\
&= \sum_{n \geq 0} (a^n : \varepsilon) && \text{by Property 2} \\
&= (a : \varepsilon) * && \text{by definition of *}
\end{aligned}$$

5. (a : epsilon)* = $a^* : \varepsilon$

Proof: same as in 4

trre normal form

Let's introduce a simplified version of TRRE language. We will call it normal form of TRRE.

Definition. Let $\mathcal{A}, \mathcal{B}$ be the alphabets. Then normal form is any string defined recursively:

1. any string of the form $a : b$ is the normal form, where $a \in \mathcal{A} \cup \{\varepsilon\}, b \in \mathcal{B} \cup \{\varepsilon\}$
2. any string of the form $A : B$, $A + B$, A^* are normal forms, where A, B are normal forms

It is clear that any normal is transductive expression. Let's show that inverse is true as well.

Lemma 1. Every trre expression can be converted to a normal form.

Proof: First, let's show that any trre expressions of the form $a : \varepsilon$ and $\varepsilon : b$ where a,b are valid regular expressions can be converted to a mormal form. Let's prove it for $A : \varepsilon$ using structural induction on the **RE** construction.

Base: $a : \varepsilon$, $\varepsilon : \varepsilon$ are in the normal form.

Step: Let's assume that all the subformulas of the formula $a : \varepsilon$ can be converted to normal form. Then let's prove that $(a + b) : \varepsilon, a \cdot b : \varepsilon, a * : \varepsilon$ can be converted to normal form:
1. $(a + b) : \varepsilon = a : \varepsilon + b : \varepsilon$ by Property 2.
2. $a \cdot b : \varepsilon = a : \varepsilon \cdot b : \varepsilon$ by Property 1.
3. $(a^* : \varepsilon) = (a : \varepsilon)^*$ by Property 4.

For the case $\varepsilon : b$ the proof is similar. So, knowing that any formula $A : B$ can be represented as $A : \varepsilon \cdot \varepsilon : B$ we can represent any TRRE of the form "A:B" in the normal form. The last step is to prove that any TRRE formula can be represented as normal form. Using current fact as a base of induction we can prove it by structural induction on the recursive definition of TRRE.

trre and automata equivalence

The final step is to show that for any TRRE there is corresponding FST which defines same language. And for any FST there is TRRE which defines same language.

Theorem.

$$\forall t \in \text{TRRE} \ \exists \ \text{FST} \ a \text{ such } \mathcal{L}(t) = \mathcal{L}(a)$$

and

$$\forall \ \text{FST} \ a \exists \ \text{TRRE} \ t \text{ such } \mathcal{L}(a) = \mathcal{L}(t)$$

Proof $\rightarrow$

The proof follows classical Thompson's construction algorithm. The only difference we represent the Transducer as an automaton over the alphabet $\mathcal{A} \cup \{\varepsilon\} \times \mathcal{B} \cup \{\varepsilon\}$.

Proof $\leftarrow$

The proof is a classical Kleene's algorithm for converting automaton to regular expression. The difference here is that we represent transductive regular expression as regular expression by first expressing TRRE in normal form (by Lemma 1) and then treating each transductive pair of the form $a : b$ as a symbol in the new alphabet. The rest of the proof is the same.

Inference

Like in many modern re engines we construct a deterministic automata on the fly. The difference here is that we construct deterministic FST whenever it is possible.