tphols-2011

By xingyuan

February 13, 2011

Contents

1 Preliminaries **1**
1.1 Finite automata and Myhill-Nerode theorem 1
1.2 The objective and the underlying intuition 3

2 Direction *regular language ⇒finite partition* **3**

3 Direction *finite partition ⇒ regular language* **6**

theory *Myhill*
imports *Myhill-2*
begin

1 Preliminaries

1.1 Finite automata and Myhill-Nerode theorem

A *determinisitc finite automata (DFA)* M is a 5-tuple $(Q, \Sigma, \delta, s, F)$, where:

1. Q is a finite set of *states*, also denoted Q_M.
2. Σ is a finite set of *alphabets*, also denoted Σ_M.
3. δ is a *transition function* of type $Q \times \Sigma \Rightarrow Q$ (a total function), also denoted δ_M.
4. $s \in Q$ is a state called *initial state*, also denoted s_M.
5. $F \subseteq Q$ is a set of states named *accepting states*, also denoted F_M.

Therefore, we have $M = (Q_M, \Sigma_M, \delta_M, s_M, F_M)$. Every DFA M can be interpreted as a function assigning states to strings, denoted $\hat{\delta}_M$, the definition of which is as the following:

$$\begin{aligned} \hat{\delta}_M([]) &\equiv s_M \\ \hat{\delta}_M(xa) &\equiv \delta_M(\hat{\delta}_M(x), a) \end{aligned} \tag{1}$$

A string x is said to be *accepted* (or *recognized*) by a DFA M if $\hat{\delta}_M(x) \in F_M$. The language recoginzed by DFA M, denoted $L(M)$, is defined as:

$$L(M) \equiv \{x \mid \hat{\delta}_M(x) \in F_M\} \tag{2}$$

The standard way of specifying a laugage $\mathcal{L}$ as *regular* is by stipulating that: $\mathcal{L} = L(M)$ for some DFA M.

For any DFA M, the DFA obtained by changing initial state to another $p \in Q_M$ is denoted M_p, which is defined as:

$$M_p \equiv (Q_M, \Sigma_M, \delta_M, p, F_M) \tag{3}$$

Two states $p, q \in Q_M$ are said to be *equivalent*, denoted $p \approx_M q$, iff.

$$L(M_p) = L(M_q) \tag{4}$$

It is obvious that $\approx_M$ is an equivalent relation over Q_M. and the partition induced by $\approx_M$ has $|Q_M|$ equivalent classes. By overloading $\approx_M$, an equivalent relation over strings can be defined:

$$x \approx_M y \equiv \hat{\delta}_M(x) \approx_M \hat{\delta}_M(y) \tag{5}$$

It can be proved that the the partition induced by $\approx_M$ also has $|Q_M|$ equivalent classes. It is also easy to show that: if $x \approx_M y$, then $x \approx_{L(M)} y$, and this means $\approx_M$ is a more refined equivalent relation than $\approx_{L(M)}$. Since partition induced by $\approx_M$ is finite, the one induced by $\approx_{L(M)}$ must also be finite, and this is one of the two directions of Myhill-Nerode theorem:

Lemma 1 (Myhill-Nerode theorem, Direction two)**.** *If a language $\mathcal{L}$ is regular (i.e. $\mathcal{L} = L(M)$ for some DFA M), then the partition induced by $\approx_{\mathcal{L}}$ is finite.*

The other direction is:

Lemma 2 (Myhill-Nerode theorem, Direction one)**.** *If the partition induced by $\approx_{\mathcal{L}}$ is finite, then $\mathcal{L}$ is regular (i.e. $\mathcal{L} = L(M)$ for some DFA M).*

The M we are seeking when prove lemma **??** can be constructed out of $\approx_{\mathcal{L}}$, denoted $M_{\mathcal{L}}$ and defined as the following:

$$Q_{M_{\mathcal{L}}} \equiv \{[\![x]\!]_{\approx_{\mathcal{L}}} \mid x \in \Sigma^*\} \tag{6a}$$
$$\Sigma_{M_{\mathcal{L}}} \equiv \Sigma_M \tag{6b}$$
$$\delta_{M_{\mathcal{L}}} \equiv (\lambda([\![x]\!]_{\approx_{\mathcal{L}}}, a).[\![xa]\!]_{\approx_{\mathcal{L}}}) \tag{6c}$$
$$s_{M_{\mathcal{L}}} \equiv [\![[]]\!]_{\approx_{\mathcal{L}}} \tag{6d}$$
$$F_{M_{\mathcal{L}}} \equiv \{[\![x]\!]_{\approx_{\mathcal{L}}} \mid x \in \mathcal{L}\} \tag{6e}$$

It can be proved that $Q_{M_{\mathcal{L}}}$ is indeed finite and $\mathcal{L} = L(M_{\mathcal{L}})$, so lemma 2 holds. It can also be proved that $M_{\mathcal{L}}$ is the minimal DFA (therefore unique) which recoginzes $\mathcal{L}$.

1.2 The objective and the underlying intuition

It is now obvious from section 1.1 that Myhill-Nerode theorem can be established easily when *reglar languages* are defined as ones recognized by finite automata. Under the context where the use of finite automata is forbiden, the situation is quite different. The theorem now has to be expressed as:

Theorem 1 (Myhill-Nerode theorem, Regular expression version)**.** *A language $\mathcal{L}$ is regular (i.e. $\mathcal{L} = L(e)$ for some* regular expression *e) iff. the partition induced by $\approx_{\mathcal{L}}$ is finite.*

The proof of this version consists of two directions (if the use of automata are not allowed):

Direction one: generating a regular expression e out of the finite partition induced by $\approx_{\mathcal{L}}$, such that $\mathcal{L} = L(e)$.

Direction two: showing the finiteness of the partition induced by $\approx_{\mathcal{L}}$, under the assmption that $\mathcal{L}$ is recognized by some regular expression e (i.e. $\mathcal{L} = L(e)$).

The development of these two directions consititutes the body of this paper.

2 Direction *regular language* $\Rightarrow$*finite partition*

Although not used explicitly, the notion of finite autotmata and its relationship with language partition, as outlined in section 1.1, still servers as important intuitive guides in the development of this paper. For example, *Direction one* follows the *Brzozowski algebraic method* used to convert finite autotmata to regular expressions, under the intuition that every partition member $[\![x]\!]_{\approx_{\mathcal{L}}}$ is a state in the DFA $M_{\mathcal{L}}$ constructed to prove lemma 2 of section 1.1.

The basic idea of Brzozowski method is to extract an equational system out of the transition relationship of the automaton in question. In the equational system, every automaton state is represented by an unknown, the solution of which is expected to be a regular expresion characterizing the state in a certain sense. There are two choices of how a automaton state can be characterized. The first is to characterize by the set of strings leading from the state in question into accepting states. The other choice is to characterize by the set of strings leading from initial state into the state in question. For the second choice, the language recognized the automaton can be characterized by the solution of initial state, while for the second choice, the language recoginzed by the automaton can be characterized by combining solutions of all accepting states by $+$. Because of the automaton

used as our intuitive guide, the $M_{\mathcal{L}}$, the states of which are sets of strings leading from initial state, the second choice is used in this paper.

Supposing the automaton in Fig 1 is the $M_{\mathcal{L}}$ for some language $\mathcal{L}$, and suppose $\Sigma = \{a, b, c, d, e\}$. Under the second choice, the equational system extracted is:

$$
\begin{aligned}
X_0 &= X_1 \cdot c + X_2 \cdot d + \lambda && \text{(7a)}\\
X_1 &= X_0 \cdot a + X_1 \cdot b + X_2 \cdot d && \text{(7b)}\\
X_2 &= X_0 \cdot b + X_1 \cdot d + X_2 \cdot a && \text{(7c)}\\
X_3 &= \begin{array}{l} X_0 \cdot (c + d + e) + X_1 \cdot (a + e) + X_2 \cdot (b + e) + \\ X_3 \cdot (a + b + c + d + e) \end{array} && \text{(7d)}
\end{aligned}
$$

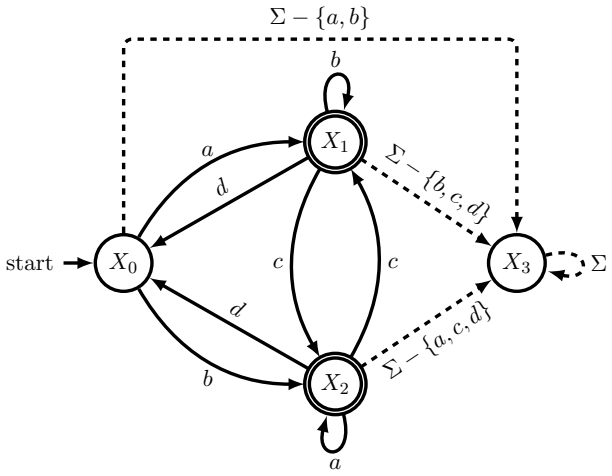

Figure 1: An example automaton

Every ·-item on the right side of equations describes some state transtions, except the λ in (7a), which represents empty string []. The reason is that: every state is characterized by the set of incoming strings leading from initial state. For non-initial state, every such string can be splitted into a prefix leading into a preceding state and a single character suffix transiting into from the preceding state. The exception happens at initial state, where the empty string is a incoming string which can not be splitted. The λ in (7a) is introduce to repsent this indivisible string. There is one and only one λ in every equational system such obtained, becasue [] can only be contaied in one equivalent class (the intial state in $M_{\mathcal{L}}$) and equivalent classes are disjoint.

Suppose all unknowns (X_0, X_1, X_2, X_3) are solvable, the regular expression charactering laugnage $\mathcal{L}$ is $X_1 + X_2$. This paper gives a procedure by which arbitrarily picked unknown can be solved. The basic idea to solve X_i is by eliminating all variables other than X_i from the equational system. If

X_0 is the one picked to be solved, variables X_1, X_2, X_3 have to be removed one by one. The order to remove does not matter as long as the remaing equations are kept valid. Suppose X_1 is the first one to remove, the action is to replace all occurences of X_1 in remaining equations by the right hand side of its characterizing equation, i.e. the $X_0 \cdot a + X_1 \cdot b + X_2 \cdot d$ in (7b). However, because of the recursive occurence of X_1, this replacement does not really removed X_1. Arden's lemma is invoked to transform recursive equations like (7b) into non-recursive ones. For example, the recursive equation (7b) is transformed into the follwing non-recursive one:

$$X_1 = (X_0 \cdot a + X_2 \cdot d) \cdot b^* = X_0 \cdot (a \cdot b^*) + X_2 \cdot (d \cdot b^*) \tag{8}$$

which, by Arden's lemma, still characterizes X_1 correctly. By subsituting $(X_0 \cdot a + X_2 \cdot d) \cdot b^*$ for all X_1 and removing (7b), we get:

$$\begin{aligned} & (X_0 \cdot (a \cdot b^*) + X_2 \cdot (d \cdot b^*)) \cdot c + X_2 \cdot d + \lambda = \\ X_0 \;=\; & X_0 \cdot (a \cdot b^* \cdot c) + X_2 \cdot (d \cdot b^* \cdot c) + X_2 \cdot d + \lambda = \\ & X_0 \cdot (a \cdot b^* \cdot c) + X_2 \cdot (d \cdot b^* \cdot c + d) + \lambda \end{aligned} \tag{9a}$$

$$\begin{aligned} & X_0 \cdot b + (X_0 \cdot (a \cdot b^*) + X_2 \cdot (d \cdot b^*)) \cdot d + X_2 \cdot a = \\ X_2 \;=\; & X_0 \cdot b + X_0 \cdot (a \cdot b^* \cdot d) + X_2 \cdot (d \cdot b^* \cdot d) + X_2 \cdot a = \\ & X_0 \cdot (b + a \cdot b^* \cdot d) + X_2 \cdot (d \cdot b^* \cdot d + a) \end{aligned} \tag{9b}$$

$$\begin{aligned} X_3 \;=\; & X_0 \cdot (c + d + e) + ((X_0 \cdot a + X_2 \cdot d) \cdot b^*) \cdot (a + e) \\ & + X_2 \cdot (b + e) + X_3 \cdot (a + b + c + d + e) \end{aligned} \tag{9c}$$

Suppose X_3 is the one to remove next, since X_3 dose not appear in either X_0 or X_2, the removal of equation (9c) changes nothing in the rest equations. Therefore, we get:

$$X_0 \;=\; X_0 \cdot (a \cdot b^* \cdot c) + X_2 \cdot (d \cdot b^* \cdot c + d) + \lambda \tag{10a}$$

$$X_2 \;=\; X_0 \cdot (b + a \cdot b^* \cdot d) + X_2 \cdot (d \cdot b^* \cdot d + a) \tag{10b}$$

Actually, since absorbing state like X_3 contributes nothing to the language $\mathcal{L}$, it could have been removed at the very beginning of this precedure without affecting the final result. Now, the last unknown to remove is X_2 and the Arden's transformaton of (10b) is:

$$X_2 \;=\; (X_0 \cdot (b + a \cdot b^* \cdot d)) \cdot (d \cdot b^* \cdot d + a)^* = X_0 \cdot ((b + a \cdot b^* \cdot d) \cdot (d \cdot b^* \cdot d + a)^*) \tag{11}$$

By substituting the right hand side of (11) into (10a), we get:

$$\begin{aligned} X_0 = \; & X_0 \cdot (a \cdot b^* \cdot c) + \\ & X_0 \cdot ((b + a \cdot b^* \cdot d) \cdot (d \cdot b^* \cdot d + a)^*) \cdot (d \cdot b^* \cdot c + d) + \lambda \\ = \; & X_0 \cdot ((a \cdot b^* \cdot c) + \\ & \quad ((b + a \cdot b^* \cdot d) \cdot (d \cdot b^* \cdot d + a)^*) \cdot (d \cdot b^* \cdot c + d)) + \lambda \end{aligned} \tag{12}$$

By applying Arden's transformation to this, we get the solution of X_0 as:

$$X_0 = ((a \cdot b^* \cdot c) + ((b + a \cdot b^* \cdot d) \cdot (d \cdot b^* \cdot d + a)^*) \cdot (d \cdot b^* \cdot c + d))^* \quad (13)$$

Using the same method, solutions for X_1 and X_2 can be obtained as well and the regular expressoin for $\mathcal{L}$ is just $X_1 + X_2$. The formalization of this procedure consititues the first direction of the *regular expression* verion of Myhill-Nerode theorem. Detailed explaination are given in **paper.pdf** and more details and comments can be found in the formal scripts.

3 Direction *finite partition* $\Rightarrow$ *regular language*

It is well known in the theory of regular languages that the existence of finite language partition amounts to the existence of a minimal automaton, i.e. the $M_{\mathcal{L}}$ constructed in section 1, which recoginzes the same language $\mathcal{L}$. The standard way to prove the existence of finite language partition is to construct a automaton out of the regular expression which recoginzes the same language, from which the existence of finite language partition follows immediately. As discussed in the introducton of **paper.pdf** as well as in [5], the problem for this approach happens when automata of sub regular expressions are combined to form the automaton of the mother regular expression, no matter what kind of representation is used, the formalization is cubersome, as said by Nipkow in [5]: '*a more abstract mathod is clearly desirable*'. In this section, an *intrinsically abstract* method is given, which completely avoid the mentioning of automata, let along any particular representations.

The main proof structure is a structural induction on regular expressions, where base cases (cases for *NULL*, *EMPTY*, *CHAR*) are quite straightforward to proof. Real difficulty lies in inductive cases. By inductive hypothesis, languages defined by sub-expressions induce finite partitiions. Under such hypothsis, we need to prove that the language defined by the composite regular expression gives rise to finite partion. The basic idea is to attach a tag $tag(x)$ to every string x. The tagging fuction *tag* is carefully devised, which returns tags made of equivalent classes of the partitions induced by subexpressoins, and therefore has a finite range. Let *Lang* be the composite language, it is proved that:

> If strings with the same tag are equivalent with respect to *Lang*, expressed as:
>
> $$tag(x) = tag(y) \Longrightarrow x \approx Lang\ y$$
>
> then the partition induced by *Lang* must be finite.

There are two arguments for this. The first goes as the following:

1. First, the tagging function *tag* induces an equivalent relation ($=tag=$) (defiintion of *f-eq-rel* and lemma *equiv-f-eq-rel*).

2. It is shown that: if the range of *tag* (denoted $range(tag)$) is finite, the partition given rise by ($=tag=$) is finite (lemma *finite-eq-f-rel*). Since tags are made from equivalent classes from component partitions, and the inductive hypothesis ensures the finiteness of these partitions, it is not difficult to prove the finiteness of $range(tag)$.

3. It is proved that if equivalent relation *R1* is more refined than *R2* (expressed as $R1 \subseteq R2$), and the partition induced by *R1* is finite, then the partition induced by *R2* is finite as well (lemma *refined-partition-finite*).

4. The injectivity assumption $tag(x) = tag(y) \Longrightarrow x \approx Lang\ y$ implies that ($=tag=$) is more refined than ($\approx Lang$).

5. Combining the points above, we have: the partition induced by language *Lang* is finite (lemma *tag-finite-imageD*).

We could have followed another approach given in appendix II of Brzozowski's paper [?], where the set of derivatives of any regular expression can be proved to be finite. Since it is easy to show that strings with same derivative are equivalent with respect to the language, then the second direction follows. We believe that our apporoach is easy to formalize, with no need to do complicated derivation calculations and countings as in [???].

end