Lecture 4: Regular Expressions and Product Construction

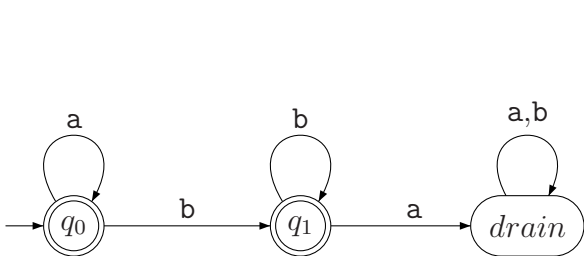
29 January 2009

This lecture finishes section 1.1 of Sipser and also covers the start of 1.3.

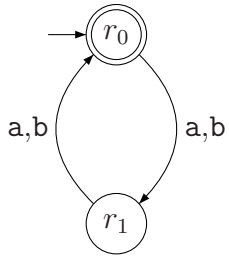
1 Product Construction

1.1 Product Construction: Example

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and L is the set of strings in Σ^* that have the form $\mathtt{a}^*\mathtt{b}^*$ and have even length. L is the intersection of two regular languages $L_1 = \mathtt{a}^*\mathtt{b}^*$ and $L_2 = (\Sigma\Sigma)^*$. We can show they are regular by exhibiting DFAs that recognize them.

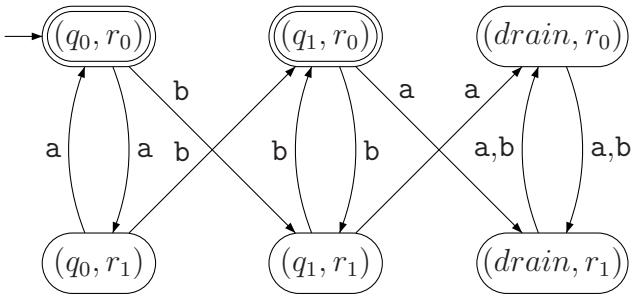


L_1



L_2

We can run these two DFAs together, by creating states that remember the states of both machines.



Notice that the final states of the new DFA are the states (q, r) where q is final in the first DFA ***and*** r is final in the second DFA. To recognize the union of the two languages, rather than the intersection, we mark all the states (q, r) such that either q or r are accepting states in the their respective DFAs.

State of a DFA after reading a word w. In the following, given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, we will be interested in what state the DFA M is in, after reading the characters of a string

$w = w_1 w_2 \ldots w_k \in \Sigma^*$. As in the definition of acceptance, we can just define the sequence of states that M would go through as it reads w. Formally, $r_0 = q_0$, and

$$r_i = \delta(r_{i-1}, w_i), \quad \text{for } i = 1, \ldots, k.$$

As such, r_k is the state M would be after reading the string w. We will denote this state by $\delta(q_0, w)$. Note, that by definition

$$\delta(q_0, w) = \delta\Big(\delta(q_0, w_1 \ldots w_{k-1}),\ w_k\Big).$$

In general, if the DFA is in a state q, and we want to know in what state it would be after reading a string w, we will denote it by $\delta(q, w)$.

2 Product Construction: Formal construction

We are given two DFAs $M = (Q, \Sigma, \delta, q_0, F)$ and $M' = (Q', \Sigma, \delta', q_0', F')$ both working above the same alphabet Σ. Their ***product automata*** is the automata

$$N = \Big(\mathcal{Q},\ \Sigma,\ \delta_N,\ (q_0, q_0'),\ F_N\Big),$$

where $\mathcal{Q} = Q \times Q'$, and $\delta_N : \mathcal{Q} \times \Sigma \to \mathcal{Q}$. Here, for $q \in Q$, $q' \in Q'$ and $c \in \Sigma$, we define

$$\delta_N(\ \underbrace{(q, q')}_{\text{state of } N}, c\ ) = \Big(\delta(q, c),\ \delta'(q', c)\Big). \tag{1}$$

The set $F_N \subseteq \mathcal{Q}$ of accepting states is free to be whatever we need it to be, depending on what we want N to recognize. For example, if we would like N to accept the intersection $L(M) \cap L(M')$ then we will set $F_N = F \times F'$. If we want N to recognize the union language $L(M) \cup L(M')$ then $F_N = (F \times Q') \ \cup\ \cup (Q \times F')$.

Lemma 2.1 *For any input word $w \in \Sigma^*$, the product automata N of the DFAs $M = (Q, \Sigma, \delta, q_0, F)$ and $M' = (Q', \Sigma, \delta', q_0', F')$, is in state (q, q') after reading w, if and only if (i) M in the state q after reading w, and (ii) M' is in the state q' after reading w.*

Proof: The proof is by induction on the length of the word w.

If $w = \epsilon$ is the empty word, then N is initially in the state (q_0, q_0') by construction, where q_0 (resp. q_0') is the initial state of M (resp. M'). As such, the claim holds in this case.

Otherwise, assume $w = w_1 w_2 \ldots w_{k-1} w_k$, and the claim is true by induction for all input words of length strictly smaller than k.

Let (q_{k-1}, q_{k-1}') be the state that N is in after reading the string $\widehat{w} = w_1 \ldots w_{k-1}$. By induction, as $|\widehat{w}| = k - 1$, we know that M is in the state q_{k-1} after reading $\widehat{w}$, and M' is in the state q_{k-1}' after reading $\widehat{w}$.

Let $q_k = \delta(q_{k-1}, w_k) = \delta(\delta(q_0, \widehat{w}),\ w_k) = \delta(q_0, w)$ and

$$q_k' = \delta'(q_{k-1}', w_k) = \delta'(\delta'(q_0', \widehat{w}),\ w_k) = \delta'(q_0', w).$$

As such, by definition, M (resp. M') would in the state q_k (resp. q_k') after reading w.

Also, by the definition of its transition function, after reading w the DFA N would be in the state

$$\begin{aligned}\delta_N((q_0, q'_0), w) &= \delta_N(\delta_N((q_0, q'_0), \widehat{w}), w_k) = \delta_N\big((q_{k-1}, q'_{k-1}),\ w_k\big) \\ &= \big(\delta(q_{k-1}, w_k),\ \delta(q'_{k-1}, w_k)\big) = (q_k,\ q'_k),\end{aligned}$$

see Eq. (1). This establishes the claim. ■

Lemma 2.2 *Let $M = (Q, \Sigma, \delta, q_0, F)$ and $M' = (Q', \Sigma, \delta', q'_0, F')$ be two given DFAs. Let N be their produced automata, where its set of accepting states is $F \times F'$. Then $L(N) = L(M) \cap L(M')$.*

Proof: If $w \in L(M) \cap L(M')$, then $q_w = \delta(q_0, w) \in F$ and $q'_w = \delta'(q'_0, w) \in F'$. By Lemma 2.1, this implies that $\delta_N((q_0, q'_0), w) = (q_w, q'_w) \in F \times F'$. Namely, N accepts the word w, implying that $w \in L(N)$, and as such $L(M) \cap L(M') \subseteq L(N)$.

Similarly, if $w \in L(N)$, then $(p_w, p'_w) = \delta_N((q_0, q'_0), w)$ must be an accepting state of N. But the set of accepting states of N is $F \times F'$. That is $(p_w, p'_w) \in F \times F'$, implying that $p_w \in F$ and $p'_w \in F'$. Now, by Lemma 2.1, we know that $\delta(q_0, w) = p_w \in F$ and $\delta'(q'_0, w) = p'_w \in F'$. Thus, M and M' both accept w, which implies that $w \in L(M)$ and $w \in L(M')$. Namely, $w \in L(M) \cap L(M')$, implying that $L(N) \subseteq L(M) \cap L(M')$.

Putting the above together implies the claim. ■

3 Operations on languages

Regular operations on languages (sets of strings). Suppose L and K are languages.

- ***Union***: $L \cup K = \left\{ x \;\middle|\; x \in L \quad \text{or} \quad x \in K \right\}$.
- ***Concatenation***: $L \circ K = LK = \left\{ xy \;\middle|\; x \in L \quad \text{and} \quad y \in K \right\}$.
- ***Star*** (***Kleene star***):

$$L^* = \left\{ w_1 w_2 \ldots w_n \;\middle|\; w_1, \ldots, w_n \in L \text{ and } n \geq 0 \right\}.$$

We (hopefully) all understand what union does. The other two have some subtleties. Let

$$L = \{\texttt{under}, \texttt{over}\}, \quad \text{and} \quad K = \{\texttt{ground}, \texttt{water}, \texttt{work}\}.$$

Then

$$LK = \{\texttt{underground}, \texttt{underwater}, \texttt{underwork}, \texttt{overground}, \texttt{overwater}, \texttt{overwork}\}.$$

Similarly,

$$K^* = \left\{ \begin{array}{l} \epsilon, \texttt{ground}, \texttt{water}, \texttt{work}, \texttt{groundground}, \\ \texttt{groundwater}, \texttt{groundwork}, \texttt{workground}, \\ \texttt{waterworkwork}, \ldots \end{array} \right\}.$$

For star operator, note that the resulting set *always* contains the empty string ϵ (because n can be zero).

Also, each of the substrings is chosen independently from the base set and you can repeat. E.g. `waterworkwork` is in K^*.

Regular languages are closed under many operations, including the three "regular operations" listed above, set intersection, set complement, string reversal, "homomorphism" (formal version of shifting alphabets). We have seen (last class) why regular languages are closed under set complement. We will prove the rest of these bit by bit over the next few lectures.

4 Regular Expressions

Regular expressions are a convenient notation to specify regular languages. We will prove in a few lectures that regular expressions can represent ***exactly*** the same languages that DFAs can accept.

Let us fix an alphabet Σ. Here are the basic regular expressions:

regex	conditions	set represented
`a`	$\mathtt{a} \in \Sigma$	$\{\mathtt{a}\}$
ϵ		$\{\epsilon\}$
$\emptyset$		$\{\}$

Thus, $\emptyset$ represents the empty language. But ϵ represents that language which has the empty word as its only word in the language.

In particular, for a regular expression $\langle \mathtt{exp} \rangle$, we will use the notation $L(\langle \mathtt{exp} \rangle)$ to denote the language associated with this regular expression. Thus,

$$L(\epsilon) = \{\epsilon\} \quad \text{and} \quad L(\emptyset) = \{\}\,,$$

which are two *different* languages.

We will slightly abuse notations, and write a regular expression $\langle \mathtt{exp} \rangle$ when in reality what we refer to is the language $L(\langle \mathtt{exp} \rangle)$. (Abusing notations should be done with care, in cases where it clarify the notations, and it is well defined. Naturally, as Humpty Dumpty did, you need to define your "abused" notations explicitly.[1])

Suppose that $L(R)$ is the language represented by the regular expression R. Here are recursive rules that make complex regular expressions out of simpler ones. (Lecture will add some randomly-chosen small concrete examples.)

[1] From *Through the Looking Glass*, by Lewis Carroll:

> 'And only one for birthday presents, you know. There's glory for you!'
>
> 'I don't know what you mean by "glory",' Alice said.
>
> Humpty Dumpty smiled contemptuously. 'Of course you don't – till I tell you. I meant "there's a nice knock-down argument for you!"'
>
> 'But "glory" doesn't mean "a nice knock-down argument",' Alice objected.
>
> 'When I use a word,' Humpty Dumpty said, in rather a scornful tone, 'it means just what I choose it to mean – neither more nor less.'
>
> 'The question is,' said Alice, 'whether you can make words mean so many different things.'
>
> 'The question is,' said Humpty Dumpty, 'which is to be master – that's all.'

regex	conditions	set represented
$R \cup S$ or $R + S$	R, S regexes	$L(R) \cup L(S)$
$R \circ S$ or RS	R, S regexes	$L(R)L(S)$
R^*	R a regex	$L(R)^*$

And some handy shorthand notation:

regex	conditions	set represented
R^+	R a regex	$L(R)L(R)^*$
Σ		Σ

Exponentiation binds most tightly, then multiplication, then addition. Just like you probably thought. Use parentheses when you want to force a different interpretation.

Some specific boundary case examples:

1. $R\epsilon = R = \epsilon R$.
2. $R\emptyset = \emptyset = \emptyset R$.

   This is a bit confusing, so let us see why this is true, recall that

   $$R\emptyset = \left\{ xy \;\middle|\; x \in R \text{ and } y \in \emptyset \right\}.$$

   But the empty set ($\emptyset$) does not contain any element, and as such, no concatenated string can be created. Namely, its the empty language.
3. $R \cup \emptyset = R$ (just like with any set).
4. $R \cup \epsilon = \epsilon \cup R$.

   This expression can not always be simplified, since ϵ might not be in the language $L(R)$.
5. $\emptyset^* = \{\epsilon\}$, since the empty word is always contain in the language generated by the star operator.
6. $\epsilon^* = \{\epsilon\}$.

4.1 More interesting examples

Suppose $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

1. $(\Sigma\Sigma)^*$ is the language of all even-length strings.

   (That is, the language associated with the regular expression $(\Sigma\Sigma)^*$ is made out of all the even-length strings over Σ.)
2. $\Sigma(\Sigma\Sigma)^*$ is all odd-length strings.
3. $\mathtt{a}\Sigma^*\mathtt{a} + \mathtt{b}\Sigma^*\mathtt{b} + \mathtt{c}\Sigma^*\mathtt{c}$ is all strings that start and end with the same character.

4.1.1 Regular expression for decimal numbers

Let $D = \{\mathtt{0}, \mathtt{1}, ..., \mathtt{9}\}$, and consider the alphabet $E = D \cup \{-, .\}$. Then decimal numbers have the form

$$(- \cup \epsilon)\, D^* \,(\epsilon \cup .) D^*.$$

But this does not force the number to contain any digits, which is probably wrong. As such, the correct expression is

$$(- \cup \epsilon)(D^+(\epsilon \cup .)D^* \cup D^*(\epsilon \cup .)D^+).$$

Notice that $\mathtt{a}^n$ is **not** a regular expression. Some things written with non-star exponents are regular and some are not. It depends on what conditions you put on n. E.g. $\left\{ \mathtt{a}^{2n} \;\middle|\; n \geq 0 \right\}$ is regular (even length strings of a's). But $\left\{ \mathtt{a}^n\mathtt{b}^n \;\middle|\; n \geq 0 \right\}$ is not regular.

However, $\mathtt{a}^3$ (or any other fixed power) is regular, as it just a shorthand for $\mathtt{aaa}$. Similarly, if R is a regular expression, then R^3 is regular since its a shorthand for RRR.