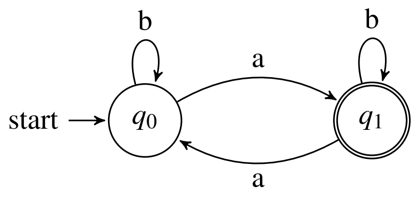
FROM FINITE AUTOMATA TO REGULAR EXPRESSIONS (USING ARDEN'S LEMMA)

Our goal is to prove the following.

Theorem 1. *Given any DFA (or NFA) M there is a regular expression E such that $L(E) = L(M)$. Furthermore there is an algorithm to construct E from M.*

The basic idea is simple, we illustrate with an example. By the way, the syntax of regular expressions varies slightly with different authors: here we use + to denote union, and we use λ to denote the empty string.

Example 2. Here's a DFA M. Note that $L(M)$ is the language of words with an odd number of as.



For each state we can write an equation which defines the set of strings taking that state to an accepting state. For example let X_0 denote the set of strings w that lead from state q_0 to an accepting state; let X_1 denote the set of strings w that lead from state q_1 to an accepting state. Then we have the following equations

$$X_0 = aX_1 + bX_0 \tag{1}$$
$$X_1 = aX_0 + bX_1 + \lambda \tag{2}$$

We include λ in the equation for A_1 precisely because X_1 is an accepting state. We do this in general: if q_i is accepting, the equation for X_i includes λ.

Note that each equation that we get from our DFA looks like $X = AX + B$ for some A which does not contain λ. A moment's thought will persuade you that this will be the case whenever we write down the equations corresponding to an automaton, even if it is non-deterministic, as long as the automaton does not have λ-transitions.

We want to solve for X_0. This is done by routine substitution just as in elementary algebra, but we need a strategy to handle equations in which the same variable occurs on both the left- and right-hand sides. This is where Arden's Lemma comes in handy.

Lemma 3 (Arden's Lemma[1]). *Let A and B be languages, and suppose that the empty string λ is not in A. Then the equation $X = AX + B$ has the unique solution $X = A^*B$.*

[1] D.N. Arden. Delayed logic and finite state machines. In Theory of Computing Machine Design, pp.1-35, U. of Michigan Press, Ann Arbor. 1960.

Before proving Arden's Lemma let us see how we can use it to compute our regular expressions.

Returning now to our example: we can rearrange (2) as

$$X_1 = bX_1 + (aX_0 + \lambda) \tag{3}$$

Now we can apply Arden's Lemma for the variable X_1 with the "A" being the language denoted by b and the "B" being the language denoted by $(aX_0 + \lambda)$. We then get

$$\begin{aligned} X_1 &= b^*(aX_0 + \lambda) \qquad (4)\\ &= b^*aX_0 + b^* \qquad (5)\end{aligned}$$

Now substitute this last expression for X_1 back into (1) and combine terms:

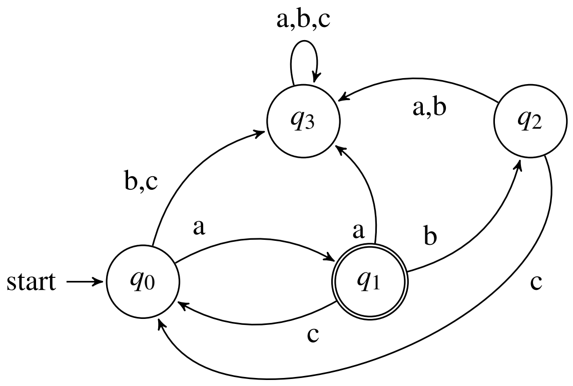
$$\begin{aligned} X_0 &= a(b^*aX_0 + b^*) + bX_0 \qquad (6)\\ &= (ab^*a)X_0 + ab^* + bX_0 \qquad (7)\\ &= (ab^*a + b)X_0 + ab^* \qquad (8)\end{aligned}$$

One more application of Arden's Lemma and we're done:

$$X_0 = (ab^*a + b)^*ab^* \tag{9}$$

Convice yourself that this regular expression really does define the set of strings with an odd number of as.

Example 4. This example is from Robin Milner's beautiful little book on concurrency: *Communicating and Mobile Systems: the* π*-calculus*.



The equations are:

$$\begin{aligned} X_0 &= aX_1 + (b+c)X_3 \qquad (10)\\ X_1 &= aX_3 + bX_2 + cX_0 + \lambda \qquad (11)\\ X_2 &= (a+b)X_3 + cX_0 \qquad (12)\\ X_3 &= (a+b+c)X_3 \qquad (13)\end{aligned}$$

Now, we can always proceed in a robotic manner but let's look around and be sensitive to some simplifications. Note that X_3 is the empty language! (You can see this by looking at the DFA, or by computing using Arden's Lemma: $X_3 = (a+b+c)^*\emptyset$, which is $\emptyset$.

Having noted that $X_3 = \emptyset$ we can simplify the equations above.

$$X_0 = aX_1 \tag{14}$$
$$X_1 = bX_2 + cX_0 + \lambda \tag{15}$$
$$X_2 = cX_0 \tag{16}$$

Now we can proceed as in the previous example. First substitute the 2nd equation into the first:

$$X_0 = a(bX_2 + cX_0 + \lambda) \tag{17}$$
$$= abX_2 + acX_0 + a \tag{18}$$

Now substitute the 3nd equation into the first:

$$X_0 = ab(cX_0) + acX_0 + a \tag{19}$$
$$= (abc + ac)X_0 + a \tag{20}$$

Now Arden gives us our answer:

$$X_0 = (abc + ac)^*a \tag{21}$$

Example 5. Another example, this time starting with an NFA. We'll just start with the equations, and not even bother to draw a picture of the NFA: if you care to you can draw it effortlessly from the equations. We do need to say that state 0 is the start state (which is to say that we want to solve for X_0).

$$\begin{aligned} X_0 &= aX_2 + aX_3 + \lambda \\ X_1 &= bX_0 \\ X_2 &= aX_1 + bX_2 \\ X_3 &= bX_0 + bX_1 + \lambda \end{aligned}$$

Note by the way that both states 0 and 3 are accepting states (you can tell because they have λ on the right-hand sides of their defining equations). First we eliminate X_1. It is not defined recursively so there is no need for Arden's lemma at this step.

$$\begin{aligned} X_0 &= aX_2 + aX_3 + \lambda \\ X_2 &= abX_0 + bX_2 \\ X_3 &= bX_0 + bbX_0 + \lambda \end{aligned}$$

Then eliminate X_3

$$\begin{aligned} X_0 &= aX_2 + a(bX_0 + bbX_0 + \lambda) + \lambda \\ X_2 &= abX_0 + bX_2 \end{aligned}$$

Now we want to eliminate X_2. We use Arden first

$$X_2 = b^*abX_0$$

Then

$$X_0 = ab^*abX_0 + a(bX_0 + bbX_0 + \lambda) + \lambda$$

Collect terms, then use Arden

$$\begin{aligned} X_0 &= (ab^*ab + ab + abb)X_0 + a + \lambda \\ &= (ab^*ab + ab + abb)^*(a + \lambda) \end{aligned}$$

So the language recognized by our machine is defined by the regular expression

$$(ab^*ab + ab + abb)^*(a + \lambda)$$

Proof of Arden's Lemma

Here is a proof of Arden's Lemma. We want to show that

> *if A is a language such that the empty string λ is not in A, then the equation $X = AX + B$ has the unique solution $X = A^*B$.*

Proof. First, it is easy to check directly that the set A^*B does satisfy the relationship $A^*B = A(A^*B) + B$. So A^*B is a solution. The interesting part is showing that this is the only solution.

So let C be any language satisfying $C = AC + B$. We want to show that in fact C must be equal to A^*B.

First note that since $AC + B \subseteq C$ we have that

$$AC \subseteq C. \tag{22}$$

and

$$B \subseteq C \tag{23}$$

But $B \subseteq C$ implies $AB \subseteq AC$, and then using $AC \subseteq C$ we get $AB \subseteq C$. Now starting with $AB \subseteq C$ we proceed in the same way: $AB \subseteq C$ so $AAB \subseteq AC \subseteq C$ and we get $AAB \subseteq C$ and so forth. In other words, for each $k \geq 0$ we can show $A^kB \subseteq C$. We conclude that $A^*B \subseteq C$.

It remains to show that $C \subseteq A^*B$. Specifically, we prove that for all w, $w \in C$ implies $w \in A^*B$, by induction on the length $|w|$ of w. Let $w \in C$. Since $C \subseteq AC + B$ we have two cases: either $w \in AC$ or $w \in B$.

In the first case, when $w \in B$, we're done, certainly $w \in A^*B$.

In the second case, $w \in AC$, we have that $w = xy$ for some $x \in A$ and $y \in C$. But now — since $\lambda \notin A$ — we know that x is not λ, and so $|y| < |w|$. So the induction hypothesis applies to y, and so y is in A^*B. So $w = xy$ is in $A(A^*B)$, and we conclude $w \in A^*B$.

Final notes

The examples here had only one accepting state but there is no complication at all if multiple state are accepting. We would simply have more than one state whose corresponding equation included λ.

Of course there can be more than one regular expression capturing $L(M)$ since different regular expressions can define the same language. This is reflected in the fact that we have strategic choices we can make as we solve the equations.

Dan Dougherty

October 14, 2009 – 16 : 42