Regular Expressions

Regular Expressions are a formal algebraic notation for specifying certain languages (regular languages, as we will show).

Def: *(Sipser, Def. 1.52, restated)* Let an alphabet Σ be fixed. The set $\mathcal{RE}$ of *regular expressions* over Σ is the smallest set of strings satisfying the following (closure) conditions:

1. $a \in \mathcal{RE}$, for each $a \in \Sigma$.
2. $\epsilon \in \mathcal{RE}$
3. $\phi \in \mathcal{RE}$
4. If $R_1 \in \mathcal{RE}$ and $R_2 \in \mathcal{RE}$, then $(R_1 \cup R_2) \in \mathcal{RE}$.
5. If $R_1 \in \mathcal{RE}$ and $R_2 \in \mathcal{RE}$, then $(R_1 \circ R_2) \in \mathcal{RE}$.
6. If $R_1 \in \mathcal{RE}$ then $(R_1{}^*) \in \mathcal{RE}$.

Notes:

- Regular expressions are *strings* over the alphabet:

$$\{\phi, \epsilon, (,), \cup, \circ, {}^{*}\} \cup \Sigma$$

(the red color emphasizes when we are referring to a symbol itself, rather than some meaning for it).

- The set $\mathcal{RE}$ has an *inductive definition*, which defines it as the smallest set with some closure properties.

- The subexpression relation $\prec$ on $\mathcal{RE}$ is *well-founded*, which implies that a *structural induction principle* is valid.

Structural Induction for Regular Expressions

Proposition (Structural Induction): Suppose $P(R)$ is a property of regular expressions such that all of the following hold:

1. $P(a)$, for all $a \in \Sigma$
2. $P(\epsilon)$
3. $P(\phi)$
4. $\forall R_1\ R_2.\ P(R_1) \wedge P(R_2) \rightarrow P((R_1 \cup R_2))$
5. $\forall R_1\ R_2.\ P(R_1) \wedge P(R_2) \rightarrow P((R_1 \circ R_2))$
6. $\forall R_1.\ P(R_1) \rightarrow P((R_1{}^*))$.

Then $P(R)$ holds for all regular expressions R.

The Language Denoted by a Regular Expression

The language $L(R)$ denoted by a regular expression R is defined (recursively) as follows:

1. $L(a) = \{a\}$, for $a \in \Sigma$.
2. $L(\phi) = \phi$ (the empty set)
3. $L(\epsilon) = \{\epsilon\}$
4. $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$.
5. $L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$.
6. $L((R_1{}^*)) = L(R_1)^*$.

(syntax on the LHS, language operations on the RHS).

Notes and Examples:

- For convenience, we omit parentheses, under the convention that $*$ has the highest precedence, then $\circ$, then finally $\cup$. We also typically drop the $\circ$ symbol, using just concatenation (as in ordinary algebra).

- We will stop using the red color unless it is necessary to clarify what are formal symbols and what is the context.

- **Example:** A regular expression R such that $L(R)$ is the set of all strings in $\{0,1\}^*$ that start and end with the same symbol:

$$R \equiv 0 \cup 1 \cup 0(0 \cup 1)^* 0 \cup 1(0 \cup 1)^* 1$$

or, fully written out (and unreadable by humans):

$$R \equiv (0 \cup (1 \cup (0(((0 \cup 1)^*)0)) \cup (1(((0 \cup 1)^*)1))))$$

Regular Algebra

Many algebraic identities hold for regular expressions which are similar to those that hold for ordinary algebra:

- $R_1 \cup R_2 = R_2 \cup R_1$
  (commutative law for $\cup$)

- $R_1 \cup (R_2 \cup R_3) = (R_1 \cup R_2) \cup R_3$
  (associative law for $\cup$)

- $R \cup \phi = R = \phi \cup R$
  (ϕ is an identity for $\cup$)

- $R \cup R = R$
  (idempotence for $\cup$)

- $R_1 \circ (R_2 \circ R_3) = (R_1 \circ R_2) \circ R_3$
  (associative law for $\circ$)
- $R \circ \epsilon = R = \epsilon \circ R$
  (ϵ is an identity for $\circ$)
- $R \circ \phi = R = \phi \circ R$
  (ϕ is a zero for $\circ$)

- $R \circ (S_1 \cup S_2) = (R \circ S_1) \cup (R \circ S_2)$
  (left distributive law of $\circ$ over $\cup$)
- $(S_1 \cup S_2) \circ R = (S_1 \circ R) \cup (S_2 \circ R)$
  (right distributive law of $\circ$ over $\cup$)

- $R^* = \epsilon \cup (R \circ R^*)$
  (left expansion for $*$)
- $R^* = \epsilon \cup (R^* \circ R)$
  (right expansion for $*$)

Verification of the Identities

Why are these laws valid?

Use the definition of $L(R)$ to verify them!

Example: $R \circ \epsilon = R$

$$\begin{aligned} L(R \circ \epsilon) &= L(R) \circ L(\epsilon) \\ &= \{xy.\ x \in L(R) \wedge y \in \{\epsilon\}\} \\ &= \{x\epsilon.\ x \in L(R)\} \\ &= \{x.\ x \in L(R)\} \\ &= L(R). \end{aligned}$$

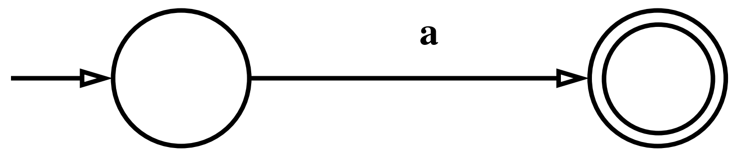
You should try to verify the others!

Lemma: *(Sipser, 1.55)* For all regular expressions R the language $L(R)$ is a regular language.

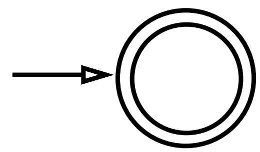
Proof: We may use Structural Induction, where $P(R)$ is "the language $L(R)$ is a regular language." We must show:

1. $L(a)$ is a regular language, for all $a \in \Sigma$.
2. $L(\epsilon)$ is a regular language.
3. $L(\phi)$ is a regular language.
4. For all R_1 and R_2, if $L(R_1)$ and $L(R_2)$ are regular languages, then $L((R_1 \cup R_2))$ is a regular language.
5. For all R_1 and R_2, if $L(R_1)$ and $L(R_2)$ are regular languages, then $L((R_1 \circ R_2))$ is a regular language.
6. For all R_1, if $L(R_1)$ is a regular language, then $L((R_1{}^*))$ is a regular language.

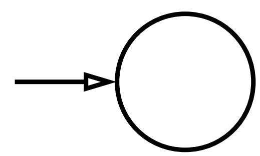
1. The following NFA recognizes $L(a)$:



2. The following NFA recognizes $L(\epsilon)$:



3. The following NFA recognizes $L(\phi)$:



4. Assume $L(R_1)$ and $L(R_2)$ are regular languages. Then $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$, which we have previously shown is regular (closure under union).

5. Assume $L(R_1)$ and $L(R_2)$ are regular languages. Then $L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$, which we have previously shown is regular (closure under concatenation).

6. Assume $L(R_1)$ is a regular language. Then $L((R_1{}^*)) = L(R_1)^*$, which we have previously shown is regular (closure under star).

An Alternative Construction

The technique just presented tends to give rather large NFAs. Using "regular expression calculus," we can often do better.

Def. If $L \subseteq \Sigma^*$ and $a \in \Sigma$, then the *derivative of L by a* is the language:

$$\partial_a L = \{x.\ ax \in L\}$$

(*i.e.* "cancel" the initial a from any strings that start with a, and discard the rest.

Prop. If $L \subseteq \Sigma^*$ is regular and $a \in \Sigma$. then $\partial_a L$ is also regular. (Proof: Exercise!)

Derivatives of Regular Expressions

Def. For $a \in \Sigma$, define $\partial_a R$, the *derivative of R by a* recursively as follows (we omit writing $\circ$, for simplicity):

1. $\partial_a b = \begin{cases} \epsilon, & \texttt{if } a = b \\ \phi, & \texttt{otherwise.} \end{cases}$
2. $\partial_a \epsilon = \phi$
3. $\partial_a \phi = \phi$
4. $\partial_a (R_1 \cup R_2) = \partial_a R_1 \cup \partial_a R_2.$
5. $\partial_a (R_1 R_2) = \begin{cases} (\partial_a R_1) R_2 \cup (\partial_a R_2), & \texttt{if } \epsilon \in L(R_1) \\ (\partial_a R_1) R_2, & \texttt{otherwise.} \end{cases}$
6. $\partial_a (R_1^*) = (\partial_a R_1) R_1^*.$

(Do you notice any similarities with differential calculus?

Prop. For all $a \in \Sigma$ and all regular expressions R:

$$L(\partial_a R) = \partial_a(L(R)).$$

Proof Idea: Use structural induction. Cases (1)-(4) are very easy; *e.g.*

$$\begin{aligned}
\partial_a(R_1 \cup R_2) &= \{x.\ ax \in L(R_1 \cup R_2)\} \\
&= \{x.\ ax \in L(R_1) \vee ax \in L(R_2)\} \\
&= \{x.\ ax \in L(R_1)\} \cup \{x.\ ax \in L(R_2)\} \\
&= L(\partial_a R_1) \cup L(\partial_a R_2).
\end{aligned}$$

For case (5):

$$
\begin{aligned}
\partial_a(R_1R_2) &= \{x.\ ax \in L(R_1R_2)\} \\
&= \{x.\ ax \in L(R_1)L(R_2)\} \\
&= \{x.\ ax \in \{yz.\ y \in L(R_1) \wedge z \in L(R_2)\}\} \\
&= \{x.\ \exists yz.\ y \in L(R_1) \wedge z \in L(R_2) \wedge yz = ax\}
\end{aligned}
$$

Consider two cases: $\epsilon \notin L(R_1)$ or $\epsilon \in L(R_1)$:

- **Case** *$\epsilon \notin L(R_1)$:* Then $y \neq \epsilon$, so $y = au$ for some $u \in \partial_a(L(R_1))$, hence $x = uz \in (\partial_a L(R_1))L(R_2) = L((\partial_a R_1)R_2)$. Thus $x \in L((\partial_a R_1)R_2)$ in this case.

- **Case** *$\epsilon \in L(R_1)$:* If $y \neq \epsilon$, then $x \in L((\partial_a R_1)R_2)$ as above. If $y = \epsilon$, then $z = ax$, so $z \in \partial_a(L(R_2)) = L(\partial_a R_2)$. Thus $x \in L((\partial_a R_1)R_2) \cup L(\partial_a R_2) = L((\partial_a R_1)R_2 \cup \partial_a R_2)$ in this case.

Determining if $\epsilon \in L(R)$

To apply this calculus, we need to be able to decide whether $\epsilon \in L(R)$. This can also be done recursively:

- $\epsilon \in L(a)$ `iff false`
- $\epsilon \in L(\epsilon)$ `iff true`
- $\epsilon \in L(\phi)$ `iff false`
- $\epsilon \in L(R_1 \cup R_2)$ `iff` $\epsilon \in L(R_1) \vee \epsilon \in L(R_2)$
- $\epsilon \in L(R_1 R_2)$ `iff` $\epsilon \in L(R_1) \wedge \epsilon \in L(R_2)$
- $\epsilon \in L(R^*)$ `iff true`.

Brzozowski's Theorem

Def. Call regular expressions R and R' *congruent* if they can be proved equal using the following algebraic laws (and the usual properties of equality):

$$\begin{aligned} R \cup R &= R \\ R_1 \cup R_2 &= R_2 \cup R_1 \\ (R_1 \cup R_2) \cup R_3 &= R_1 \cup (R_2 \cup R_3) \end{aligned}$$

Theorem *(Brzozowski, 1964):* Every regular expression R has at most finitely many non-congruent derivatives. These form the set of states of a DFA that recognizes $L(R)$.

Proof: (You'll have to read Brzozowski's paper!)

A Refinement

Although for termination only the three laws listed above are required, in practice it is useful to use additional laws; namely:

$$
\begin{aligned}
R \cup \phi &= R \\
R\phi &= \phi \\
\phi R &= \phi \\
R\epsilon &= R \\
\epsilon R &= R
\end{aligned}
$$

We will use these additional laws, which results in fewer states.

Obtaining a DFA from a Regular Expression

Brzozowski's Theorem can be used to take a regular expression R and obtain an equivalent DFA:

- *Initialize* Q with $Q = \{R\}$.
- *Repeat* until no new element E of Q remains to be examined (termination by Brzozowski's Theorem):
  - *Calculate* $\partial_a E$ for each $a \in \Sigma$ and add the result to Q if it is not congruent to any existing element of Q.
- *Define* $q_0 = R$ and take F to be the set of all $E \in Q$ such that $\epsilon \in L(E)$.
- *Define* $\delta : Q \times \Sigma \to Q$ by $\delta(E, a) = \hat{E}$, where $\hat{E} \in Q$ is congruent to $\partial_a E$.

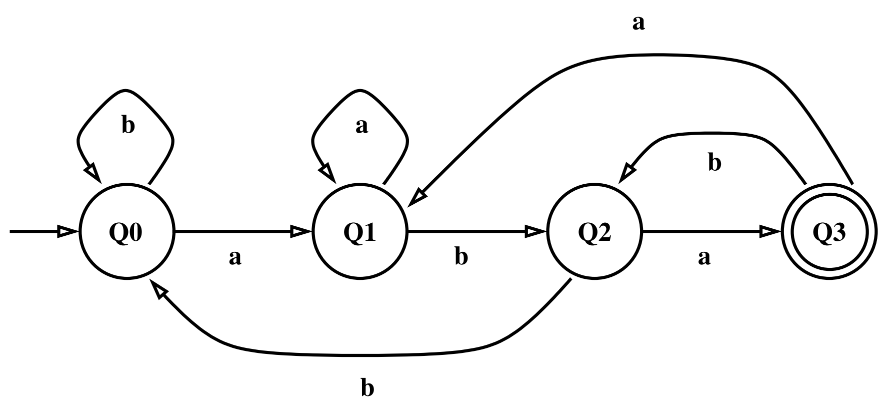
Example

Let's do Sipser Example 1.58 ($R = (a \cup b)^* aba$) using this technique:

- $q_0 = (a \cup b)^* aba$
- Calculate:

$$\begin{aligned}
\partial_a q_0 &= \partial_a((a \cup b)^* aba) \\
&= (\partial_a (a \cup b)^*) aba \cup \partial_a(aba) \\
&= (\epsilon (a \cup b)^*) aba \cup ba \\
&= (a \cup b)^* aba \cup ba \\
&= q_1
\end{aligned}$$

- $\partial_b q_0 = \partial_b((a \cup b)^* aba) = (a \cup b)^* aba = q_0$
- $\partial_a q_1 = \partial_a((a \cup b)^* aba \cup ba) = (a \cup b)^* aba \cup ba = q_1$
- $\partial_b q_1 = \partial_b((a \cup b)^* aba \cup ba) = (a \cup b)^* aba \cup a = q_2$
- $\partial_a q_2 = \partial_a((a \cup b)^* aba \cup a) = (a \cup b)^* aba \cup ba \cup \epsilon = q_3$
- $\partial_b q_2 = \partial_b((a \cup b)^* aba \cup a) = (a \cup b)^* aba = q_0$
- $\partial_a q_3 = \partial_a((a \cup b)^* aba \cup ba \cup \epsilon) = (a \cup b)^* aba \cup ba = q_1$
- $\partial_b q_3 = \partial_b((a \cup b)^* aba \cup ba \cup \epsilon) = (a \cup b)^* aba \cup a = q_2$



Constructing a Regular Expression from a NFA

Given an NFA it is possible to construct an equivalent regular expression.

Lemma (Kleene, 1956): *(Sipser, 1.55)* If L is a regular language, then $L = L(R)$ for some regular expression R.

The basic idea is an "elimination procedure" that works by eliminating states. Sipser expresses this in terms of *generalized nondeterministic finite automata*, which are NFA's whose transitions are labeled by regular expressions.

Generalized NFA's

Def: *(Sipser, 1.64)* A *generalized nondeterministic finite automaton* is a 5-tuple

$$(Q, \Sigma, \delta, q_{\texttt{start}}, q_{\texttt{accept}}),$$

where

1. Q is the finite set of states.
2. Σ is the input alphabet.
3. $\delta : (Q - \{q_{\texttt{accept}}\}) \times (Q - \{q_{\texttt{start}}\}) \rightarrow \mathcal{RE}$ is the transition function.
4. $q_{\texttt{start}}$ is the start state.
5. $q_{\texttt{accept}}$ is the accept state.

Acceptance by a GNFA

Def. A GNFA *accepts* a string $w \in \Sigma^*$ if w can be written $w = w_1 w_2 \ldots w_k$, where each $w_i \in \Sigma^*$, and there exists $q_0, q_1, \ldots, q_k$ such that

1. $q_0 = q_{\mathtt{start}}$,
2. $q_k = q_{\mathtt{accept}}$, and
3. For each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

That is, when we take a transition from q_{i-1} to q_i, we consume input w_i such that $w_i \in L(R_i)$, where R_i is the label of the transition.

Notes

- Sipser seems to assume $q_{\texttt{start}} \neq q_{\texttt{accept}}$, so $|Q| \geq 2$ (but the definition does not imply this).
- The start state $q_{\texttt{start}}$ is a "source" (no incoming transitions).
- The accept state $q_{\texttt{accept}}$ is a "sink" (no outgoing transitions).
- In all cases for (q, r) other than (- , $q_{\texttt{start}}$) and ($q_{\texttt{accept}}$, -), δ assigns a unique label to (q, r).
- A 2-state GNFA has $Q = \{q_{\texttt{start}}, q_{\texttt{accept}}\}$ and recognizes language $L(R)$, where $R = \delta(q_{\texttt{start}}, q_{\texttt{accept}})$.

An Ordinary NFA Determines an Equivalent GNFA

Given an ordinary NFA, we can obtain an equivalent GNFA:

- Add $q_{\texttt{start}}$ and $q_{\texttt{accept}}$ as new states.
- Set $\delta(q_{\texttt{start}}, q_0) = \epsilon$, and $\delta(q_{\texttt{start}}, q_i) = \phi$ for $i \neq 0$.
- Set $\delta(q_j, q_{\texttt{accept}}) = \epsilon$ for $q_j \in F$ and $\delta(q_j, q_{\texttt{accept}}) = \phi$ for $q_j \notin F$.
- Eliminate multiple transitions between the same pair of states (q_i, q_j), by replacing them with a single transition labeled by a union.
- If (q_i, q_j) originally had no transition, then set $\delta(q_i, q_j) = \phi$.

Procedure for Converting a GNFA to a Regular Expression

- If the GNFA has exactly 2 states, return $\delta(q_{\texttt{start}}, q_{\texttt{accept}})$.
- If the GNFA has > 2 states, choose a state $q_{\texttt{rip}}$ to be eliminated, where $q_{\texttt{rip}} \notin \{q_{\texttt{start}}, q_{\texttt{accept}}\}$.
  - Obtain Q' by removing $q_{\texttt{rip}}$ from Q.
  - Define δ' so that

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

    where $R_1 = \delta(q_i, q_{\texttt{rip}})$, $R_2 = \delta(q_{\texttt{rip}}, q_{\texttt{rip}})$, $R_3 = \delta(q_{\texttt{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

Correctness of the Conversion

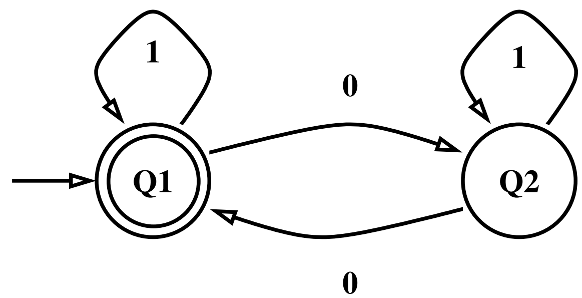
Key Idea: *For all $q_i \neq q_{\mathtt{rip}}$ and $q_j \neq q_{\mathtt{rip}}$, the set of strings w that could be read along some path from q_i to q_j* ***before*** *removal is the same as the set that can be read along some such path* ***afterwards****.*

- *Before removal*, each path from q_i to q_j either *avoids $q_{\mathtt{rip}}$ entirely* or else goes *from q_i to $q_{\mathtt{rip}}$* for the first time, then *from $q_{\mathtt{rip}}$ to $q_{\mathtt{rip}}$* zero or more times, then *from $q_{\mathtt{rip}}$* for the last time *to q_j*.
  - Every string w that can be read *before removal* along a path from q_i to q_j *that avoids $q_{\mathtt{rip}}$* entirely can still be read along the same path after removal (due to the $\cup\ (R_4)$ terms).

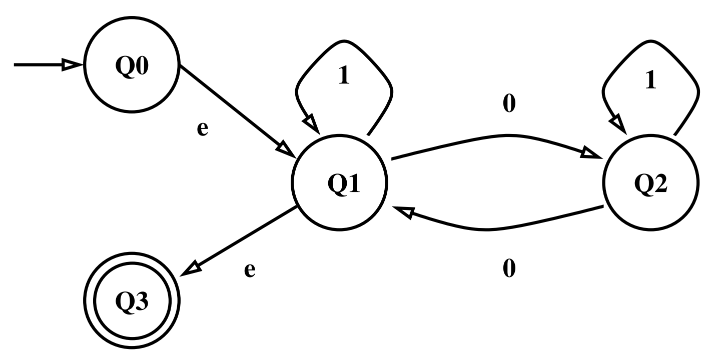
- If w could be read *before removal* along a path from q_i to q_j *that visits $q_{\mathtt{rip}}$*, then $w \in L((R_1)(R_2)^*(R_3))$, so after removal it can be read along the direct path from q_i to q_j.
- If w can be read *after removal* along some path from q_i to q_j, then it could also have been read along some such path *beforehand* (this is true for single steps, hence extends to all paths by induction).

Example

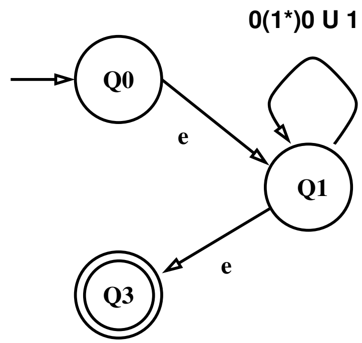
Consider the DFA:



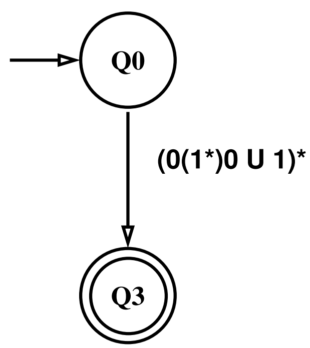
Convert to a GNFA:



Remove $Q2$:



Remove $Q1$:



NFA's as Systems of Equations

Another way to understand the previous construction is in terms of systems of equations: Suppose N is a NFA, with $Q = \{q_0, q_1, \ldots, q_n\}$ For each j with $0 \leq j \leq n$, write an equation:

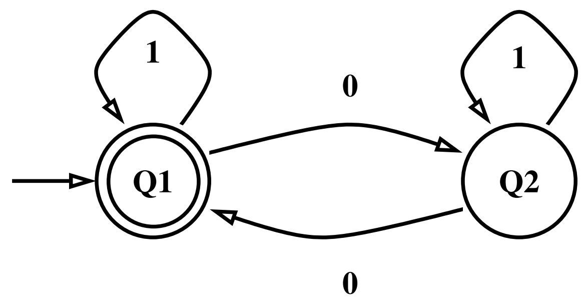
$$X_i = (\bigcup_{j=0}^{n} D_j \circ X_j) \cup C_i$$

where the X_i are *variables* and C_j and D_i are *regular expressions*, given by:

- $D_j = \bigcup \{y.\ q_j \in \delta(q_i, y)\}$
- $C_i = \begin{cases} \epsilon, & \texttt{if } q_i \in F \\ \phi, & \texttt{if } q_i \notin F \end{cases}$

Example

Consider the DFA:



$$\begin{aligned} X_1 &= (1 \circ X_1) \cup (0 \circ X_2) \cup \epsilon \\ X_2 &= (0 \circ X_1) \cup (1 \circ X_2) \end{aligned}$$

Solution to Equations

Def. Regular expressions R_0, R_1, . . . , R_n *solve* an equation

$$X_j = \mathcal{E}(X_0, X_1, \ldots, X_n)$$

if

$$L(R_j) = L(\mathcal{E}(R_0, R_1 \ldots, R_n)).$$

That is, "plugging in" R_i for X_i yields an equation that is true for languages.

Prop. *Suppose R_0, R_1, . . . , R_n solve the equations formed from an NFA N. Then $L(R_i)$ is the set of all strings w that are accepted by N starting in state q_i.*

Proof omitted.

Solving Equations by Elimination

We can solve the equations formed from an NFA by an *elimination procedure*, roughly analogous to solving ordinary systems of linear equations.

Arden's Lemma: A solution to the equation

$$X = D \circ X \cup C$$

is given by $X = D^* \circ C$. Moreover, if $\epsilon \notin L(D)$, then this solution is unique, in the sense that if R and S are any two solutions, then $L(R) = L(S)$.

We can use Arden's Lemma to successively eliminate all variables except the one for the start state. Solving the one remaining equation gives a regular expression R such that $L(R)$ is the language recognized by N.

Example

Consider again the set of equations for the example NFA:

$$\begin{aligned} X_1 &= (1 \circ X_1) \cup (0 \circ X_2) \cup \epsilon \\ X_2 &= (0 \circ X_1) \cup (1 \circ X_2) \end{aligned}$$

To eliminate X_2, use Arden's Lemma to solve the second equation:

$$X_2 = 1^* \circ (0 \circ X_1)$$

Then substitute this solution into the first equation:

$$X_1 = (1 \circ X_1) \cup (0 \circ 1^* \circ (0 \circ X_1)) \cup \epsilon$$

Rewrite (using regular algebra identities) into form $D \circ X_1 \cup C$:

$$X_1 \;=\; (1 \cup 0 \circ 1^* \circ 0) \circ X_1 \cup \epsilon$$

Solve for X_1 using Arden's Lemma again and simplify:

$$X_1 = (1 \cup 0 \circ 1^* \circ 0)^* \circ \epsilon = (1 \cup 0 \circ 1^* \circ 0)^*$$

Read off the regular expression for $L(N)$:

$$(1 \cup 0 \circ 1^* \circ 0)^*$$