DFA to RE

Seetha Abhinav, Piyush Kumar, Aarav Desai, Abhinav Goyal
UG BTech 2nd Year

5 APRIL 2025

Outline

- Review of Kleene's theorem
- Top Down Elimination
- Arden's Lemma

And much more!.....

Kleene's theorem:1

Theorem (Kleene)

Every regular language over Σ can be constructed from $\emptyset$ and $\{a\}$, $a \in \Sigma$, using only the operations union, concatenation and Kleene star.

Regular languages are closed under other operations such as intersection and complement, but these are not needed to construct a regular language from the basic ones.

Before we sketch the proof, let us introduce the notation system for regular languages suggested by Kleene's result.

Kleene's Theorem:2

Regular Expressions

Definition

A regular expression is a term constructed as follows:

- Basic expressions: $\emptyset$, a for all $a \in \Sigma$
- Operators: $(E_1 + E_2)$, $(E_1 \cdot E_2)$, (E^*)

For example,

$$((a \cdot (b^*)) + c)$$

is a regular expression. While correct, this is too clumsy for words: as usual in arithmetic, one uses precedence ordering to avoid parentheses and drops the dot for concatenation.

One often allows ε as a primitive denoting the empty word (though this is technically redundant since $\emptyset^*$ denotes the same language).

Kleene's Theorem:3

The Corresponding Languages

We can associate a language with each regular expression:

$$\mathcal{L}(\emptyset) = \emptyset$$

$$\mathcal{L}(a) = \{a\}$$

$$\mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$$

$$\mathcal{L}(E_1 \cdot E_2) = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2)$$

$$\mathcal{L}(E^*) = \mathcal{L}(E)^*$$

So by Kleene's theorem, for every regular language L there is a regular expression α such that $\mathcal{L}(\alpha) = L$.
Lastly, one avoids the underlines and writes things like $ab^* + c$, it's always clear from context what is meant. One should also be relaxed about identifying $\{x\}$ and x whenever convenient.

Kleene's Theorem: 4

RE Example

Example

- All words containing bab: $(a+b)^*bab(a+b)^*$
- All words containing $3a's$: $b^*ab^*ab^*ab^*$
- All words not containing aaa: $(\varepsilon + a + aa)(b + ba + baa)^*$

Exercise

- Construct a regex for all words with an even number of $a's$ and $b's$.

Exercise

- Construct a regex for all words with an odd number of $a's$ and $b's$.

Kleene's Theorem:5

Proof Sketch Kleene

One direction is easy given the results with already have.
To show that $\mathcal{L}(\alpha)$ is regular for α all we need is induction on the build-up of α:

- The claim is trivial for atomic regular expressions.
- For compound regular expressions use closure under union, concatenation and Kleene star.

More on realistic implementations later.

Kleen's Theorem:6

Suppose we have an NFA that accepts some regular language L. Assume

$$Q = [n].$$

For p, q in Q define

$$L_{p,q} = \mathcal{L}(\langle Q, \Sigma, \delta; \{p\}, \{q\}\rangle)$$

Then

$$L = \bigcup_{p \in I, q \in F} L_{p,q}$$

and it suffices to construct regular expressions for the $L_{p,q}$.
In order to obtain an inductive argument, define a run from state p to state q to be k-bounded if all intermediate states are no greater than k. Note that p and q themselves are not required to be bounded by k.
Now consider the approximation languages:

$$L^k_{p,q} = \{x \in \Sigma^* \mid \text{there is a } k\text{-bounded run } p \xrightarrow{x} q\}.$$

Kleene's Theorem:7

Think of the NFA as a building with n floors:

- **States** are floors $(1, 2, 3, \ldots, n)$.
- **Transitions** are elevators (labeled by symbols from Σ).
- A path from floor p to q represents a string taking the NFA from state p to q.

k-bounded run:

- You can move between floors, but intermediate stops must be $\leq k$.
- Example: If $k = 2$, you can use floors 1 and 2, but not 3 or higher.

Building the Language Step-by-Step:

- Start with only the lowest floors $(k = 1)$.
- Gradually allow higher floors $(k = 2, k = 3, \ldots)$.
- Each step builds on the previous, ensuring the language remains regular.

Kleene's Theorem:8

Proof Sketch, cont'd.

One can build expressions for $L^k_{p,q}$ by induction on k.
For $k = 0$ the expressions are easy:

$$L^0_{p,q} = \begin{cases} \sum_{\tau(p,a,q)} a & \text{if } q \neq p, \\ \sum_{\tau(p,a,q)} a + \varepsilon & \text{otherwise.} \end{cases}$$

So suppose $k > 0$. The key idea is to use the equality

$$L^k_{p,q} = L^{k-1}_{p,q} + L^{k-1}_{p,k} \cdot (L^{k-1}_{k,k})^* \cdot L^{k-1}_{k,q}$$

Done by IH.
Note that

$$L^n_{p,q} = L_{p,q}.$$

Kleene's Theorem:9

- **Base Case** ($k = 0$):
  - Direct transitions with **no intermediate states**
  - Regular expressions for $L^0_{p,q}$ are either:
    - Union of edge labels $a \in \Sigma$ for direct $p \to q$ transitions
    - ϵ if $p = q$ (empty path)

- **Inductive Step** ($k \to k+1$):
  - Paths either avoid state $k+1$ or use it as intermediate:

$$L^{k+1}_{p,q} = L^k_{p,q} \cup \left(L^k_{p,k+1} \cdot \left(L^k_{k+1,k+1} \right)^* \cdot L^k_{k+1,q} \right)$$

  - Where:
    - $L^k_{p,q}$: Existing paths avoiding $k+1$
    - $L^k_{p,k+1}$: Path to the new state
    - $\left(L^k_{k+1,k+1} \right)^*$: Loops at $k+1$
    - $L^k_{k+1,q}$: Path from $k+1$ to q

Kleene's Theorem: 10

Pseudo Code

```
foreach p = 1,...,n do
    foreach q = 1,...,n do
        initialize A[p,q,0];
foreach k = 1,...,n do
    foreach p = 1,...,n do
        foreach q = 1,...,n do
            A[p,q,k] = A[p,q,k-1] +
A[p,k,k-1].A[k,k,k-1]*.A[k,q,k-1];
return sum( A[p,q,n] | p in I, q in F );
```

Exercise:

- Analyze the structure of the loops in the above pseudocode.
- What is the time complexity of this algorithm? Justify your answer.
- Compare this with Floyd-Warshall's shortest path algorithm. What similarities do you notice?

Kleene's Theorem:11

Time Complexity Analysis

- **Triply nested loops** dominate the computation:
  - Outer loop: $k = 1, \ldots, n$
    (intermediate states)
  - Inner loops: $p = 1, \ldots, n$ and $q = 1, \ldots, n$
    (all state pairs)

- Total iterations:

$$\underbrace{n}_{\text{outer}} \times \underbrace{n}_{\text{middle}} \times \underbrace{n}_{\text{inner}} = \mathcal{O}(n^3)$$

- **Operations per iteration**:
  - Union $(+)$, concatenation $(\cdot)$, and Kleene star $(^*)$
  - Abstract model assumes constant-time regex operations

Practical Consideration

While the *algorithmic pattern* is $\mathcal{O}(n^3)$, actual regex size grows exponentially due to Kleene star operations.

Machine to RE

There are three basic approaches to converting a finite state machine to a regular expression:

- Kleene's State elimination method
  The proof of Kleene's theorem provides a dynamic programming algorithm.
- Linear systems of equations
  The machine is converted into a system of linear equations over the language semiring. The system can then be solved using Arden's lemma.

Converting a finite state machine to a regular expression is a bit of an academic exercise. The key problem is that for all approaches to yield manageable expressions one needs to simplify the intermediate results. There are heuristics to do that, but in general simplification is computationally hard.

Why Learn Other Methods?

The proof of Kleene's theorem provides a direct dynamic programming algorithm: construct expression $\alpha(p, q, k)$ in stages $k = 0, \ldots, n$ for $1 \leq p, q \leq n$, assuming the state set is $[n]$.
There are two problems this algorithm:

- There are $n^2(n+1)$ expressions to construct, which is bad but not fatal.
- The expressions roughly quadruple in size in the step from level k to level $k+1$. This is a disaster.

In practice, one has to simplify the expressions to make them smaller and choose the elimination order cleverly so that some of the expressions are not needed (top-down versus bottom-up dynamic programming).

Top Down Elimination:1

Here is a reasonable method for small machines.

- Add two new states b (begin) and e (exit) to the machine. Add e-transitions from b to all initial states, and from all final states to e.
- Think of the edge labels as regular expressions. We will successively remove all states other than b and e.
- To this end pick some state $r \in Q$. For all incoming transitions $p \xrightarrow{\alpha} r$, outgoing transitions $r \xrightarrow{\gamma} q$, and self-loops $r \xrightarrow{\beta} r$, add a new transition

$$p \xrightarrow{\alpha\beta^*\gamma} q$$

  In the end, remove r and all incident transitions.
- Repeat until only b and e remain.

Top Down Elimination: 2

Result

After all states in Q are eliminated, we are left with

$$b \xrightarrow{\alpha} e$$

where α denotes the language of the machine.

The proof is to show by induction that, at any point during construction, the 'automaton' is equivalent to the original. Here we need to generalize the notion of an automaton a little: Allow regular expressions to be treated as labels rather than just letters.

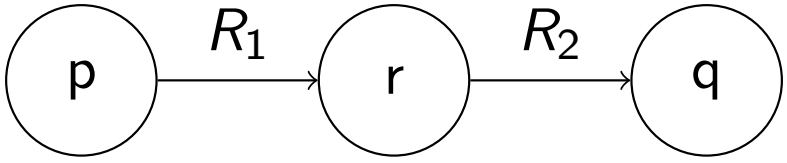
Exercise

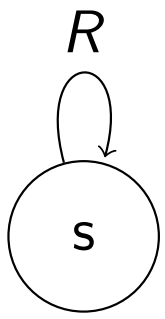
- **Proof strategy**: Formalize the inductive argument:
  - **Base case**: Show equivalence after adding b and e with ε-transitions
  - **Inductive step**: Assume equivalence after k eliminations. Prove that eliminating the $(k+1)^{th}$ state preserves equivalence using:

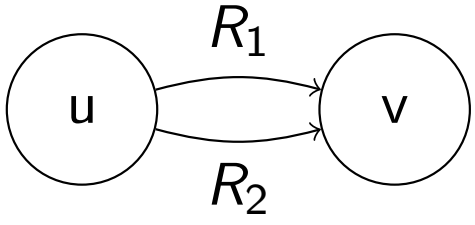
$$p \xrightarrow{\alpha\beta^*\gamma} q \text{ replaces all paths through } r$$

Top Down Elimination:3

- **Concatenation:** Sequence of R_1 followed by R_2 is $R_1 R_2$.
- **Kleene Star:** Loop on R is R^*, meaning repeat R zero or more times.
- **Union:** Choice between R_1 and R_2 is $R_1|R_2$.

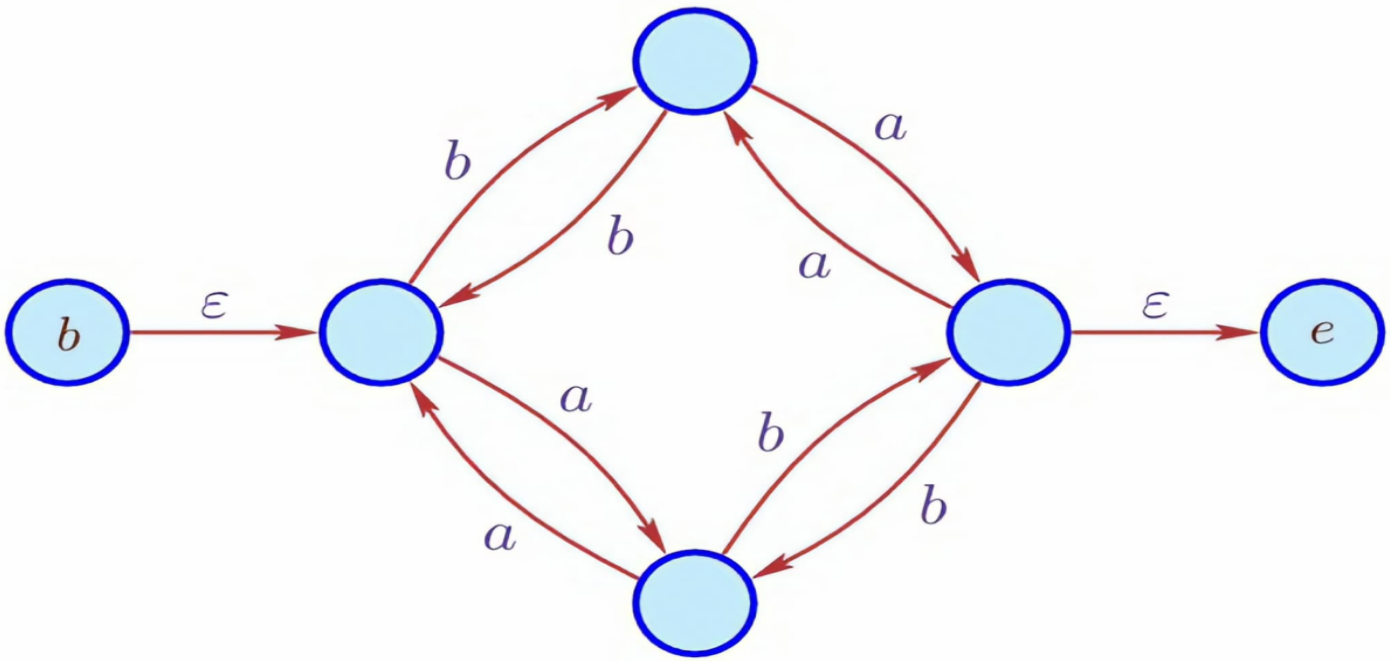






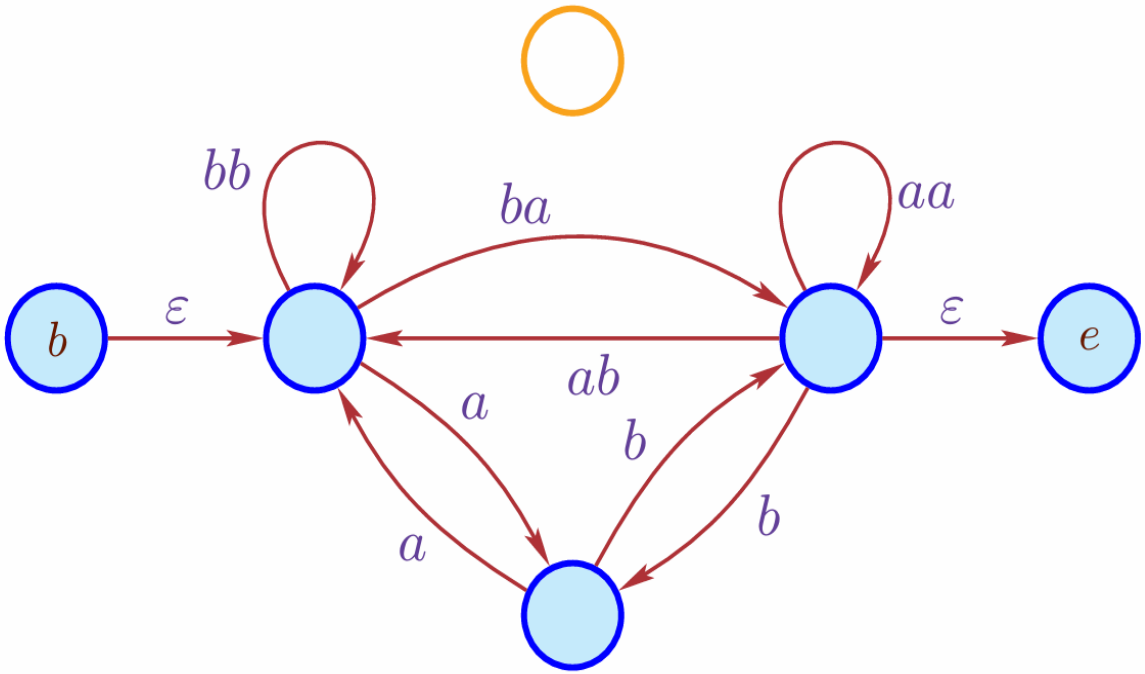
Top Down Elimination:4

Odd/Odd, Step 0



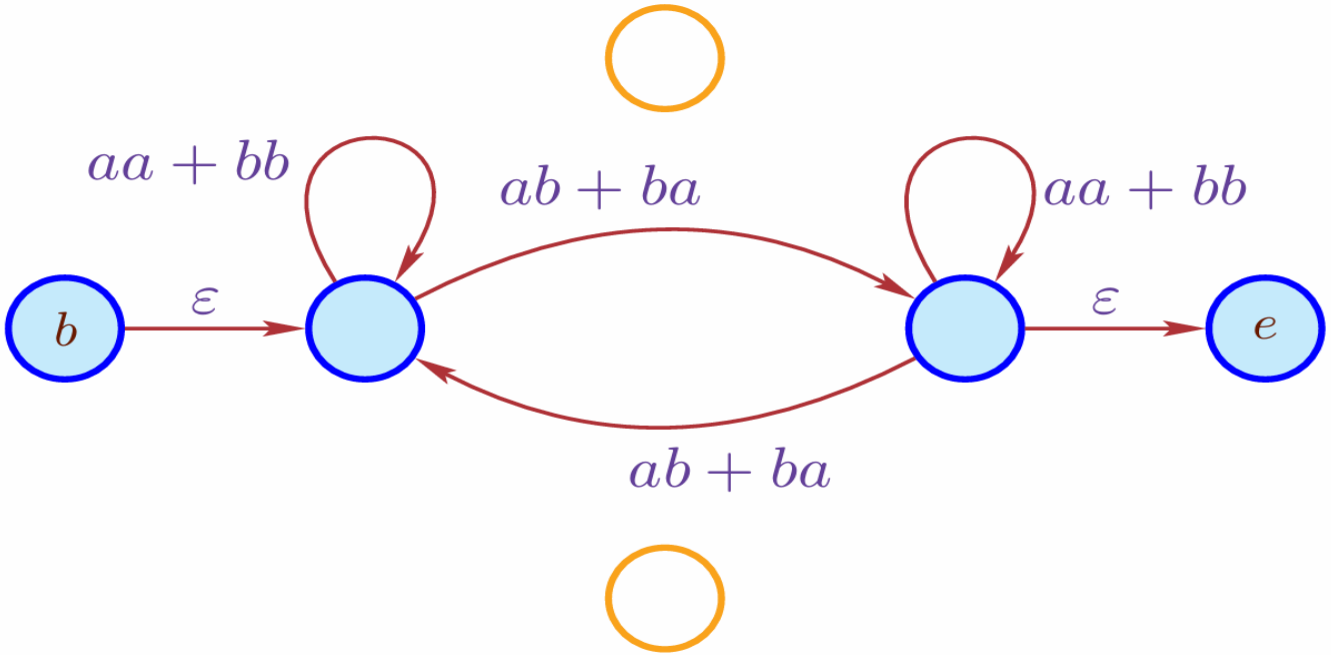
Top Down Elimination:5

Odd/Odd, Step 1



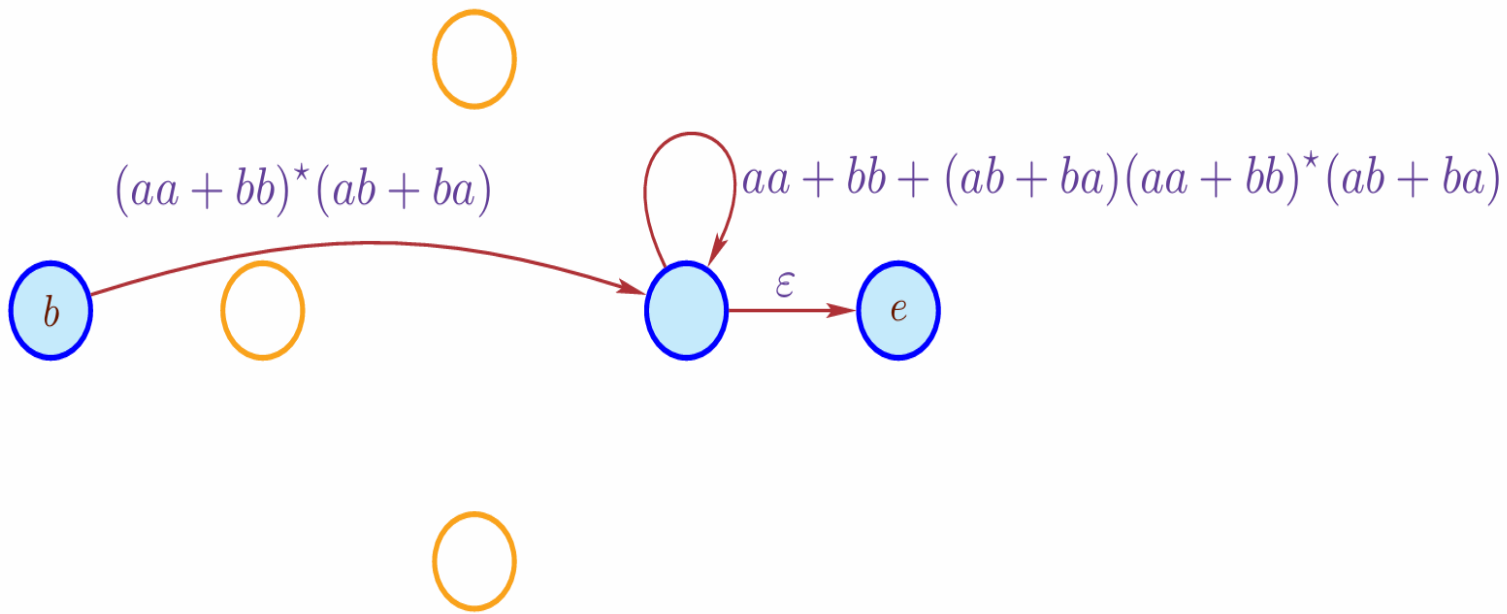
Top Down Elimination:6

Odd/Odd, Step 2



Top Down Elimination:7

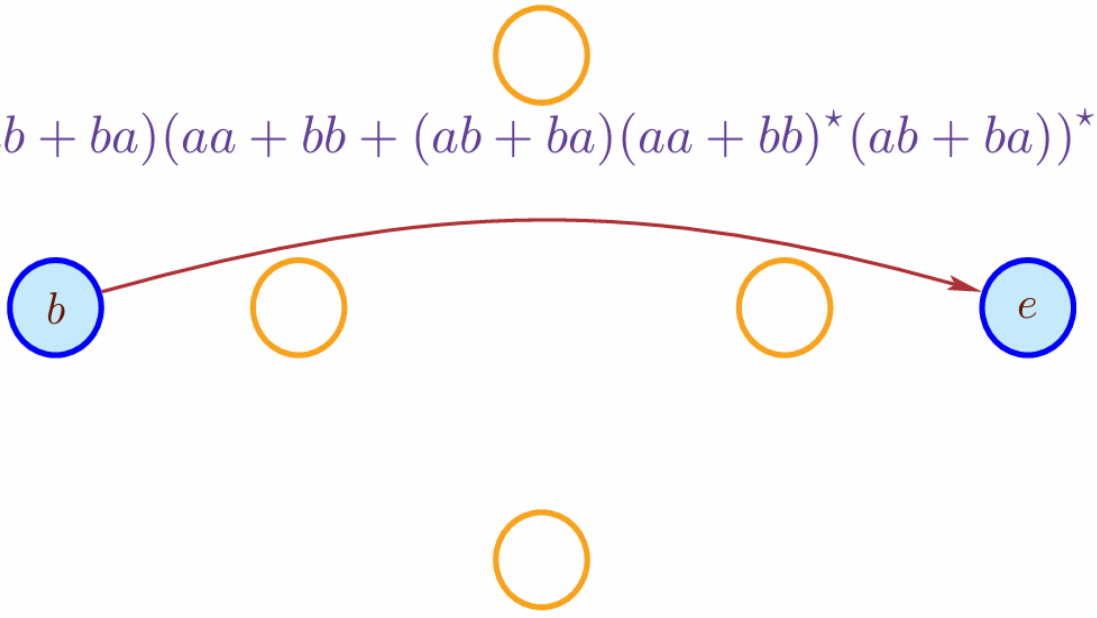
Odd/Odd, Step 3



Top Down Elimination:8

Odd/Odd, Step 4

$$(aa + bb)^\star(ab + ba)(aa + bb + (ab + ba)(aa + bb)^\star(ab + ba))^\star$$



Top Down elimination:9

The final regex is

$$\alpha = (aa + bb)^*(ab + ba)(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$$

It's not completely clear that this is correct.
A little argument shows that the last part

$$(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$$

denotes all even/even words.
Then α must be correct, too: all odd/odd words consist of a (possibly empty) prefix $(aa + bb)^*$, followed by $(ab + ba)$, followed by an even/even word.

To keep the size of the expressions small it is important to apply various simplifications. For example, the following rules seem reasonable:

$$\emptyset \cdot \alpha \mapsto \emptyset$$

$$\varepsilon \cdot \alpha \mapsto \alpha$$

$$\alpha + \alpha \mapsto \alpha$$

$$\varepsilon + \alpha\alpha^* \mapsto \alpha^*$$

Unfortunately, the algebra of regular expressions is complicated and there is no simple set of rules that would produce reasonable expressions.

review of some basic operations

Basic Equations

Disregarding Kleene star, the basic rules are not too bad:

$$(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$$

$$\alpha + \beta = \beta + \alpha$$

$$\emptyset + \alpha = \alpha$$

$$\alpha + \alpha = \alpha$$

$$(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$$

$$\varepsilon \cdot \alpha = \alpha \cdot \varepsilon = \alpha$$

$$\emptyset \cdot \alpha = \alpha \cdot \emptyset = \emptyset$$

$$\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$$

$$(\beta + \gamma) \cdot \alpha = \beta \cdot \alpha + \gamma \cdot \alpha$$

Review of some basic operations

Rules for kleen

But Kleene star causes huge problems. Here are some (natural?) rules:

$$(\alpha + \beta)^* = (\alpha^* \beta)^* \alpha^*$$

$$(\alpha\beta)^* = \epsilon + \alpha(\beta\alpha)^* \beta$$

$$(\alpha^*) = \alpha^*$$

$$\alpha^* = (\epsilon + \alpha + \ldots + \alpha^{n-1}) \cdot (\alpha^n)^*$$

Note that the last equation is actually an infinite family of equations; it is known that no finite family will do.
Not to mention that for simplification we need rewrite rules, not equations.

Frightening case

Here is the result running a conversion algorithm with some degree of simplifications on the even/even example (which is easier than odd/odd):

$$\begin{aligned}\epsilon + b(bb)^*b + (a + b(bb)^*ba)(aa + ab(bb)^*ba)^*(a + ab(bb)^*b) + \\ (b(bb)^*a + (a + b(bb)^*ba)(aa + ab(bb)^*ba)^*(b + ab(bb)^*a)) \\ (a(bb)^*a + (b + a(bb)^*ba)(aa + ab(bb)^*ba)^*(b + ab(bb)^*a))^* \\ (a(bb)^*b + (b + a(bb)^*ba)(aa + ab(bb)^*ba)^*(a + ab(bb)^*b))\end{aligned}$$

It takes quite a bit of effort just to check that this expression describes the even/even language.

Arden's Lemma: 1

Consider a linear language equation of the form

$$X = A \cdot X + B.$$

where $A, B \subseteq \Sigma^*$ are arbitrary languages, though we will be mostly interested in the case where A and B are finite or perhaps regular.
We are looking for a solution $X_0 \subseteq \Sigma^*$.
As it turns out, linear equations are rather easy to solve.

Lemma (Arden's Lemma)

Let A and B be languages over Σ. Then the equation $X = A \cdot X + B$ has a solution $X_0 = A^*B$. Moreover, if $\epsilon \notin A$, then this solution is unique. In any case, X_0 it is the smallest solution (with respect to set-theoretic inclusion).

Exercise

proof of Arden's Lemma?

Arden's Lemma: 2

Verify X = A*B is a solution (part 1)

We must show:

$$A(A^*B) \cup B = A^*B$$

$$A(A^*B) = A\left(\bigcup_{k\geq 0} A^k B\right)$$

$$= \bigcup_{k\geq 0} A^{k+1} B \quad \text{(Distributive property)}$$

Key observations:

- $B = A^0 B$ (since $A^0 = \{\epsilon\}$)
- $\bigcup_{k\geq 0} A^{k+1} B = \bigcup_{k\geq 1} A^k B$

Arden's Lemma: 3

Verify X = A*B is a solution (part 2)

Combining the results:

$$\begin{aligned} A(A^*B) \cup B &= \left(\bigcup_{k \geq 1} A^k B \right) \cup (A^0 B) \\ &= \bigcup_{k \geq 0} A^k B \\ &= A^* B \end{aligned}$$

Thus we've shown:

$$A(A^*B) \cup B = A^*B$$

which means $X_0 = A^*B$ satisfies the equation $X = AX \cup B$.

Arden's Lemma: 4

Show $A^*B \subseteq Z$ (*Part*1)

Let Z satisfy:

$$Z = AZ \cup B$$

Prove by induction that $\forall k \geq 0$:

$$A^k B \subseteq Z$$

Base Case (k=0):

$$A^0 B = \{\epsilon\} B = B$$

$$\text{Since } Z = AZ \cup B \Rightarrow B \subseteq Z$$

Inductive Step:

Assume $A^k B \subseteq Z$ for some $k \geq 0$. Need to show:

$$A^{k+1} B \subseteq Z$$

Arden's Lemma: 5

Show $A^*B \subseteq Z$ (*Part*2)

Inductive Step Continued:

$$\begin{aligned} A^{k+1}B &= A(A^kB) \\ &\subseteq AZ \quad \text{(by inductive hypothesis } A^kB \subseteq Z\text{)} \\ &\subseteq Z \quad \text{(since } Z = AZ \cup B\text{)} \end{aligned}$$

Conclusion:

$$\forall k \geq 0,\ A^kB \subseteq Z \Rightarrow A^*B = \bigcup_{k \geq 0} A^kB \subseteq Z$$

Thus $X_0 = A^*B$ is the **smallest solution**.

Arden's Lemma: 6

Uniqueness When $\epsilon \notin A$ (Part 1)

Claim: If Z is any solution of $Z = AZ \cup B$, and if $\epsilon \notin A$, then every $x \in Z$ can be written as $x = a_1 a_2 \cdots a_n b$, with each $a_i \in A$ (none of the a_i are empty) and $b \in B$; that is, $x \in A^* B$.

Proof (by induction on the length $|x|$):

Base Case: Let $x \in Z$ with the minimal possible length.

If $x \in B$, then since $A^0 = \{\epsilon\}$ and $A^0 B = B$, we have $x \in A^* B$.

Otherwise, if x were produced by the recursive part, then $x \in AZ$. However, because $\epsilon \notin A$, every element $a \in A$ has $|a| \geq 1$. Thus, if $x = ay$ with $a \in A$ and $y \in Z$, then $|x| = |a| + |y| > |y|$. This contradicts the minimality of $|x|$ unless $x \in B$. Hence, the base case forces $x \in B \subseteq A^* B$.

Inductive Step: Assume that for all strings $y \in Z$ with $|y| < n$, we have $y \in A^* B$. Now take any $x \in Z$ with $|x| = n$.

Arden's Lemma: 7

Uniqueness When $\epsilon \notin A$ (Part 2)

Since $Z = AZ \cup B$ and x is not a minimal element (if it were in B, we are done), there must exist $a \in A$ and $y \in Z$ such that $x = ay$. Because $\epsilon \notin A$, we have $|a| \geq 1$ so that $|y| < |x|$. By the induction hypothesis, $y \in A^*B$, so we can write $y = a_2 a_3 \cdots a_n b$, where each $a_i \in A$ and $b \in B$. Then:

$$x = a(a_2 a_3 \cdots a_n b) = a a_2 a_3 \cdots a_n b,$$

which shows $x \in A^*B$.
Since every $x \in Z$ is obtained by a finite number of concatenations of nonempty strings from A (followed by a string from B), we conclude $Z \subseteq A^*B$. But from our earlier induction (Step 2 of the overall proof), we had $A^*B \subseteq Z$. Therefore, when $\epsilon \notin A$,

$$Z = A^*B.$$

This completes the uniqueness proof for the case $\epsilon \notin A$.

Arden's Lemma: 8

Non-Uniqueness When $\epsilon \in A$ (I)

Claim: If $\epsilon \in A$, solutions $X = AX \cup B$ are non-unique.

Mathematical Reasoning:

- Decompose $A = A' \cup \{\epsilon\}$ where A' contains nonempty strings
- Equation becomes:

$$X = A'X \cup \{\epsilon\}X \cup B = X \cup A'X \cup B$$

- Equivalent to $X \supseteq A'X \cup B$ (no contraction)
- Any X containing B and closed under A' operations satisfies equality

Lack of Contraction:

- $\epsilon \in A$ allows $x \mapsto \epsilon x = x$
- No strict length increase $\Rightarrow$ no inductive uniqueness

Arden's Lemma: 9

Non-Uniqueness When $\epsilon \in A$ (II)

Example:

$$A = \{\epsilon\},\ B = \{b\} \Rightarrow X = X \cup \{b\}$$

- Minimal solution: $A^*B = \{\epsilon\}^*\{b\} = \{b\}$
- Non-minimal solution: $Y = \{b, a\}$ satisfies:

$$Y = Y \cup \{b\} \quad (\text{since } b \in Y)$$

General Case:

- Any superset $Y \supset A^*B$ satisfies $Y = Y \cup A'Y \cup B$
- $\epsilon \in A$ makes $AX = X$ for any X
- Infinitely many solutions exist beyond A^*B

Arden's Lemma: 10

Regularity

Note that Arden's lemma implies that equation $X = A \cdot X + B$ has a regular solution $X_0 = A^*B$ whenever A and B are regular. Moreover, if A does not contain ε this is the only solution. If A and B are given as rational expressions we obtain a rational expression for the solution.

Example

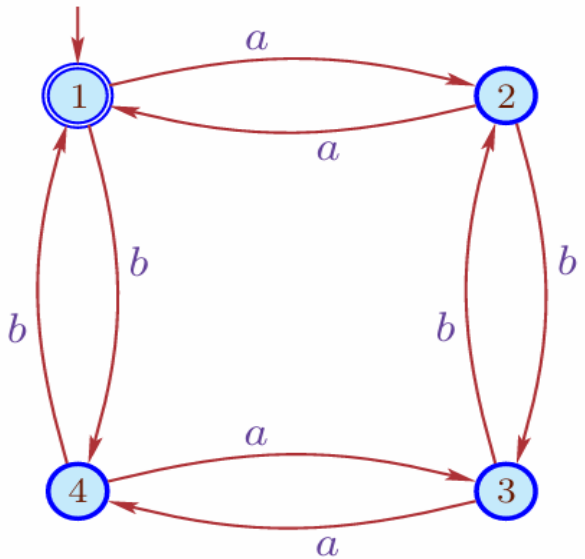
The equation $X = bX + a$ has b^*a as its unique solution.

By contrast, the equation $X = b^*X + a$ has infinitely many solutions $X_k = b^*(a + a^2 + \cdots + a^k)$ and $X_1 = b^*a$ is the least solution. Indeed, there are uncountably many solutions $b^* \cdot L$ where $a \in L \subseteq a^*$ is arbitrary.

Arden's Lemma: 11

Equations:

Let's return to the even/even language. The canonical DFA looks like so:



We convert the DFA into a system of equations.

Arden's Lemma: 12

Corresponding System

$$X_1 = \varepsilon + aX_2 + bX_4 \tag{1}$$

$$X_2 = aX_1 + bX_3 \tag{2}$$

$$X_3 = aX_4 + bX_2 \tag{3}$$

$$X_4 = aX_3 + bX_1 \tag{4}$$

Substituting (2) and (4) into (1) and (3) we get

$$X_1 = \varepsilon + (aa + bb)X_1 + (ab + ba)X_3 \tag{5}$$

$$X_3 = (aa + bb)X_3 + (ab + ba)X_1 \tag{6}$$

Applying Arden's lemma to (6) we get

$$X_3 = (aa + bb)^*(ab + ba)X_1 \tag{7}$$

Arden's Lemma: 13

The Solution

Substituting (7) into (5)

$$X_1 = \varepsilon + ((aa + bb) + (ab + ba)(aa + bb)^*(ab + ba))X_1$$

Applying Arden's lemma one more time we get

$$X_1 = (aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$$

which solution makes intuitive sense.
It is important to note, though, that we tacitly did quite a bit of cleanup along the way, writing the expressions in a simplified form. Remember the horror example, from above.

Arden's Lemma: 14

Another Example

Consider the system

$$X = aX + bY$$

$$Y = \varepsilon + aX + ba^*$$

If we solve for X in the first equation, $X = a^*bY$, and substitute in the second and then solve for Y and resubstitute we get

$$X = a^*b(a^+b)^*(\varepsilon + ba^*)$$

$$Y = (a^+b)^*(\varepsilon + ba^*)$$

So far, so good.

Arden's Lemma: 15

A Different Approach

However, we could also first substitute the second equation into the first and solve for X.
This leads to solutions

$$X' = (a + b(ab)^* aa)^* b(ab)^* (\varepsilon + ba^*)$$

$$Y' = (ab)^* \left(aa(a + b(ab)^* aa)^* b(ab)^* (\varepsilon + ba^*) + \varepsilon + ba^*\right)$$

Unless we made a mistake, these expressions must be equivalent to X and Y, but this is certainly not obvious from looking at them. What is sorely missing here is a simplification algorithm that brings a regular expression into a normal form. We can check for equivalence by converting to finite state machines and then testing these for equivalence.

Arden's Lemma: 16

Matrix Method

The even/even system from above could also be written as

$$\vec{X} = A \cdot \vec{X} + B$$

where

$$A = \begin{pmatrix} 0 & a & 0 & b \\ a & 0 & b & 0 \\ 0 & b & 0 & a \\ b & 0 & a & 0 \end{pmatrix}$$

and $B = (1, 0, 0, 0)$.
Here we have written 0 and 1 instead of $\emptyset$ and $\{\varepsilon\}$ for legibility.

One can then define the Kleene star of A

$$A^* = I + A + A^2 + \ldots + A^n + \ldots$$

and it is not hard to see that the solution is of the form

$$X = A^* B$$

Of course, to make this truly useful we need to find a way to compute A^*. In simple cases one can determine an approximation $\sum_{i \leq n} A^i$ for some small n and then make an educated guess as to what A^* might be.

A better way is to take a closer look at the underlying algebraic structure:

$$\mathcal{L}(\Sigma) = \langle \mathfrak{P}(\Sigma^*), \cup, \cdot, *, \emptyset, \varepsilon \rangle$$

This is one example of a closed semiring: intuitively, a semiring that also supports an infinite sum operation:

$$x^* = \sum_{i \geq 0} x^i = x^0 + x^1 + x^2 + \ldots$$

Of course, this is a little problematic: the "..." don't really have any precise meaning.

Arden's Lemma: 19

Closed Semiring Axioms

Let S be an idempotent semiring with an additional infinitary operation

$$\sum_{i \in I} a_i .$$

S is a closed semiring if this infinite summation operation behaves properly with respect to finite sums, distributivity and reordering of the arguments. More precisely, we require

$$\sum_{i \in [n]} a_i = a_1 + \ldots + a_n,$$

$$\left(\sum_{i \in I} a_i \right) \left(\sum_{j \in J} b_j \right) = \sum_{(i,j) \in I \times J} a_i b_j,$$

$$\sum_{i \in I} a_i = \sum_{j \in J} \left(\sum_{i \in I_j} a_i \right)$$

Arden's Lemma: 20

Consequences

Note the arbitrary index sets floating around in these axioms. They are inherently more complicated than traditional, purely equational axioms for standard structures such as groups, rings or fields.
At any rate, one can still derive general results from these axioms:
Lemma In any closed semiring we have

$$(x + y)^* = (x^* y)^* x^*$$

$$(xy)^* = 1 + x(yx)^* y$$

It is easy to check that $\mathcal{L}(\Sigma)$ is a closed semiring. More importantly, the collection of all $n \times n$ matrices over $\mathcal{L}(\Sigma)$ is again a closed semiring:

$$\mathcal{L}(\Sigma)^{n\times n} = \langle \Re(\Sigma^*)^{n\times n}, \cup, \cdot, *, 0, 1\rangle$$

where 0 denotes the null matrix (all entries $\emptyset$) and 1 denotes the identity matrix ($\{\varepsilon\}$ along the diagonal). The additive operation is simply pointwise, and multiplication is standard matrix multiplication. Star is defined as above by an infinite sum.

Arden's Lemma :22

Computing Star

So far all we have is a clean framework. Algorithmically, we still need to find a way to calculate a star in $\mathcal{L}(\Sigma)^{n\times n}$.
To this end consider a matrix

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

where the pieces have size $n/2$ (put floors and ceilings in the right places).
Then there is a divide-and-conquer algorithm to compute X^*.

Arden's Lemma: 23

Computing Star II

Let $Y = (A + BD^*C)^*$ and $Z = (D + CA^*B)^*$. Then

$$X^* = \left(\begin{array}{c|c} Y & YBD^* \\ \hline ZCA^* & Z \end{array}\right)$$

Since the dimension of the component matrices shrinks by $1/2$ there are only $\log n$ levels in the recursion. Of course, the expressions can still get ugly.

Arden's lemma : 24

Example

Let's compute the star of

$$X = \begin{pmatrix} 0 & a & 0 & b \\ a & 0 & b & 0 \\ 0 & b & 0 & a \\ b & 0 & a & 0 \end{pmatrix}$$

So the submatrices are

$$A = D = \begin{pmatrix} 0 & a \\ a & 0 \end{pmatrix} \quad B = C = \begin{pmatrix} 0 & b \\ b & 0 \end{pmatrix}$$

A moment's thought shows

$$\begin{pmatrix} 0 & x \\ x & 0 \end{pmatrix}^* = \begin{pmatrix} (xx)^* & x(xx)^* \\ x(xx)^* & (xx)^* \end{pmatrix}$$

Arden's Lemma: 25

Example II

But then

$$Y = Z = (A + BA^*B)^* = \begin{pmatrix} b(aa)^*b & a + ba(aa)^*b \\ a + ba(aa)^*b & b(aa)^*b \end{pmatrix}^*$$

Since we are only interested in the top/left entry of X^* (as $B = (1, 0, 0, 0)$ in the above example; see Arden's lemma :16), we can apply the decomposition one more time to obtain yet another regular expression for the even/even language:

$$X^*(1,1) = (b(aa)^*b + (a + ba(aa)^*b)(b(aa)^*b)^*(a + ba(aa)^*b))^*$$

Arden's Lemma: 26

Three Expressions

We now have two different but equivalent expressions for the even/even language, obtained by different methods:

$$(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$$

$$(b(aa)^*b + (a + ba(aa)^*b)(b(aa)^*b)^*(a + ba(aa)^*b))^*$$

Exercise

Make sure you understand how these three expressions work.

References

The End