Regular Expressions $\leftrightarrow$ Languages

1. KLEENE Theorem  $e \rightarrow A_e$ (DFA)
2. Reversed direction  $A_e \rightarrow e$

Focus: DFA

$S_i \rightarrow S^A_i$

transition function as matrix

Data Structure for representing DFAs
then treat matrix as regex.

How do we calculate the * of matrix?

$e$ vector for indicating final states.
classic proof: state elimination for proving DFA \(\rightarrow\) regex

At least 3 states to start with:

![Diagram showing initial states and transitions](image)

- Starting state and final state

If deleting a state, send the transition related to that state should be transitioned elsewhere. See below:

![Diagram showing deleted state and new transitions](image)

Starting from the left most and come back to the final state, we can use the following regex:

\[
(a+b(b^*)a^*)(a+b^*)
\]

If the automaton has more than one final state, we can use \(\varepsilon\) transition on all final states and have them transition to one final state.

![Diagram showing transitions and final state](image)

Then we can keep doing the state-elimination until there are only 2 states left.

Below are the 4 possibilities of the final states:

![Diagram showing final state configurations](image)
Two regexes are equal \((e_1 \equiv e_2)\) iff the two DFAs are equal.

KLEENE wondered if there were a set of equations that can decide syntactically whether 2 regexes are equal. KLEENE Algebra: \(K, 0, 1, +, *, (\cdot)^*\).

A set \(k\) that satisfies the following laws:

1. \(e + e = e\)
2. \(e + f = f + e\)
3. \((e + f) + g = e + (f + g)\)
4. \(e + 0 = e\)

\[
(e; f); g \equiv e;(f; g) \quad \text{monoid}
\]
\[
e; 1 \equiv 1; e
\]
\[
e; 0 \equiv 0; e
\]
\[
e;(f + g) \equiv e;f + e;g
\]
\[
(e + f); g \equiv e;g + f;g
\]

\(\text{semi-ring}\)

\(e^* \equiv 1 + e; e^*\)

\(e^* \equiv 1 + e^*; e\)

natural order on semiring

\(e = f \Rightarrow e + f \equiv f\)

partial order
Axion \( \{ e^x + f \leq x \} \) \& iteratively replace \( x \) with

Scheme \( e^x ; f \leq x \rightarrow (e^x ; x + f) \) to get \( e^x ; f \)

\( e^x \) is a Least Fix Point. \( f \) is the stop branch. "+" is like branching: with the left branch, recursion continues; with the right branch, it stops.

Exercises for this lecture:
1. \( x^x . x^x \equiv x^x \)
2. \( x^x \equiv (x^x)^* \)
3. \( x . y = y^2 \rightarrow x^x . y \equiv y . z^x \)
4. \( (a + b)^* \equiv (a^* \cdot b^*)^* \cdot a^* \) (denesting rule)

Examples for Kleene Algebra:

\( \langle 2^*, \emptyset, \{ \varepsilon \}, \cup, \cdot, (-)^* \rangle \)\n\( \uparrow \) Regular language. \( \uparrow \) Concatenation.

\( \langle \text{BRel}, \emptyset, \Delta, \cup, o, (-)^* \rangle \)\n\( \uparrow \) Binary Relation \( \uparrow \) Composition \( \uparrow \) Transitive closure

\( \langle \text{MAT}(K), 0, 1, \uparrow, X, (\cdot)^* \rangle \)\n\( \uparrow \) Pointwise plus
\[[a\ b]^* = \begin{bmatrix} (a+bd^*)^* & (a+bd^*)^* & bd^* \\ c & d \end{bmatrix} \]

is the matrix star for the following DFA:

\[
\begin{array}{ccc}
\emptyset & b & \emptyset \\
\\Rightarrow & & \Rightarrow \\
\emptyset & d & \emptyset
\end{array}
\]

How do we use the matrix above for calculating the star of matrix of any size? See breaking down of a square matrix of size 3 below:

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

From Kozen '93:

\[
\text{[e]} = \text{[f]} \iff e = \mathcal{K} + f
\]

\(\text{soundness}\)

\(\text{completeness}\)
To prove $e \equiv f$

Thompson construction

$Ae \quad E \quad Af$

def\downarrow \quad \downarrow\min\min$

$Ae' \quad Af'$

$Me \equiv Mf$

$$e \xrightarrow{\text{KLEENE Algebra Axioms}} Me$$

Do this syntactically for every arrow in the diagram.

This is helpful for verifying simple imperative programs.

If $b$ then $P$ else $q$

$$b \lor b_1$$

$$b \land \overline{b}_1$$

we need to combine kleene algebra and boolean algebra: $(B \leq K$, or more precisely $B$ is a sub-algebra of $K)$

$$(K, 0, 1, +, ;, (-)^*) (B, 0, 1, +, ;, ()^*) \text{in BA}$$
$2^A^* = \{ p, q, r, s, t, b_0, b_1, \ldots, b_n \}
\{ \overline{b_0}, \overline{b_1}, \ldots, \overline{b_n} \}$

what should not $(\rightarrow)$ satisfy?

$\cdot \overline{a + b} = \overline{a} \cdot \overline{b} \n\cdot \overline{a \cdot b} = \overline{a} + \overline{b} \n\cdot \overline{0} = 1 \n\cdot \overline{a \cdot b} = b \cdot \overline{a}; a \text{ conjunction is commutative} \n\cdot \overline{a} = a \n$

if $b$ then $p$ else $q \rightarrow b \cdot p + \overline{b} \cdot q$ while $b$ do $p \rightarrow (b \cdot p)^* \cdot \overline{b}$
Assumption: one final state

Any automaton ending one extra state + ε₀

eliminate until 2st.