

A new Computation System

- DFAs, NFs, Λ -NFAs all describe a language in an operational manner. They describe a machine that one can execute in order to recognize a string.
- An alternate method of describing a set of strings is to describe the properties it should have.
- Regular-Expressions are based upon the closure properties we have studied.

Regular Expressions

- Fix an alphabet Σ . We define now regular expressions and how each of them specifies a language.
- **Definition.** The set of *regular expressions* (with respect to Σ) is defined inductively by the following rules:
 1. The symbols \emptyset and Λ are regular expressions
 2. Every symbol $\alpha \in \Sigma$ is a regular expression
 3. If E and F are regular expressions, then (E^*) , (EF) and $(E+F)$ are regular expressions.

Juxtaposition of two RE uses an implicit \bullet or concatenation

Note how the closure properties are used here

Computation system as Data

- We have made a big point that computation systems are just data, regular expressions are no exception.
- We can represent them as data. Here we use Haskell as an example.

```
data RegExp a
  = Lambda
  | Empty
  | One a
  | Union (RegExp a) (RegExp a)
  | Cat (RegExp a) (RegExp a)
  | Star (RegExp a)
```

- How would you represent regular expressions in your favorite language.

Regular Expressions as Languages

- **Definition.** For every regular expression E , there is an associated language $L(E)$, defined inductively as follows:

1. $L(\emptyset) = \emptyset$ and $L(\Lambda) = \{\Lambda\}$

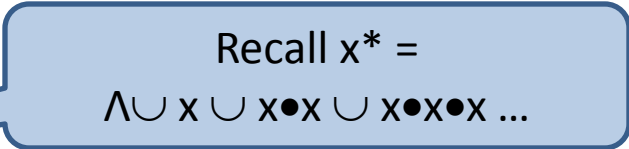
2. $L(a) = \{a\}$

3. Inductive cases

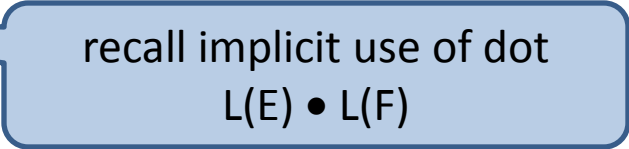
1. $L(E^*) = (L(E))^*$

2. $L(EF) = L(E) L(F)$

3. $L(E+F) = L(E) \cup L(F)$



Recall $x^* =$
 $\Lambda \cup x \cup x \bullet x \cup x \bullet x \bullet x \dots$



recall implicit use of dot
 $L(E) \bullet L(F)$

- **Definition.** A language is *regular* if it is of the form $L(E)$ for some regular expression E .

Equivalence

1. We say that regular expressions E and F are *equivalent* iff $L(E)=L(F)$.
2. We treat equivalent expressions as equal (just as we do with arithmetic expressions; (e.g., $5+7 = 7+5$)).
3. Equivalences $(E+F)+G = E+(F+G)$ and $(EF)G=E(FG)$ allow us to omit many parentheses when writing regular expressions.
4. Even more parentheses can be omitted when we declare the precedence ordering of the three operators :
 1. star (binds tightest)
 2. concatenation
 3. union (binds least of all)

RE's over $\{0,1\}$

- Fill in the blank

- $E_1 = 0+11$ then $L(E_1) = \underline{\hspace{2cm}}$

- $E_2 = (00+01+10+11)^*$ then $L(E_2) = \underline{\hspace{2cm}}$

- $E_3 = 0^*+1^*$ then $L(E_3) = \underline{\hspace{2cm}}$

- $E_4 = (00^*+11^*)^*$ then $L(E_4) = \underline{\hspace{2cm}}$

- $E_5 = (1+\Lambda)(01)^*(0+\Lambda)$ then $L(E_5) = \underline{\hspace{2cm}}$

-

Computing a language

- We can compute a language by using the definition of the meaning of a regular expression
- $L(a+b.c^*) = L(a) \cup L(bc^*)$
- $L(a+b.c^*) = L(a) \cup (L(b).L(c^*))$
- $L(a+b.c^*) = \{a\} \cup (\{b\} . \{c\}^*)$
- $L(a+b.c^*) = \{a\} \cup (\{b\} . \{\Lambda, c, cc, ccc, \dots, c^n\})$
- $L(a+b.c^*) = \{a\} \cup (\{b, bc, bcc, bccc, \dots, bc^n\})$
- $L(a+b.c^*) = \{a, b, bc, bcc, bccc, \dots, bc^n\}$

Laws about Regular expressions

- The regular expressions form an algebra
- There are many laws (just as there are laws about arithmetic $(5+2)=(2+5)$)

Laws about +

1. $R + T = T + R$

2. $R + \emptyset = \emptyset + R = R$

3. $R + R = R$

4. $R + (S + T) = (R + S) + T$

Laws about .

1. $R.\emptyset = \emptyset.R = \emptyset$

2. $R.\Lambda = \Lambda.R = R$

3. $(R.S).T = R.(S.T)$

• With Implicit .

1. $R\emptyset = \emptyset R = \emptyset$

2. $R\Lambda = \Lambda R = R$

3. $(RS)T = R(ST)$

Distributive Properties

1. $R(S + T) = RS + RT$

2. $(S + T)R = SR + TR$

Closure Properties *

1. $\emptyset^* = \Lambda^* = \Lambda$
2. $R^* = R^*R^* = (R^*)^* = R + R^*$
3. $R^* = \Lambda + R^* = (\Lambda + R)^* = (\Lambda + R)R^* = \Lambda + RR^*$
4. $R^* = (\Lambda + \dots + R^k)^*$ for all $k \geq 1$
5. $R^* = \Lambda + R + \dots + R^{(k-1)} + R^kR^*$ for all $k \geq 1$
6. $RR^* = R^*R$
7. $R(SR)^* = (RS)^*R$
8. $(R^*S)^* = \Lambda + (R + S)^*S$
9. $(RS^*)^* = \Lambda + R(R + S)^*$

Next time

- We will study how to make recognizers from regular expressions
- We will prove that RE and DFAs describe the same class of languages.