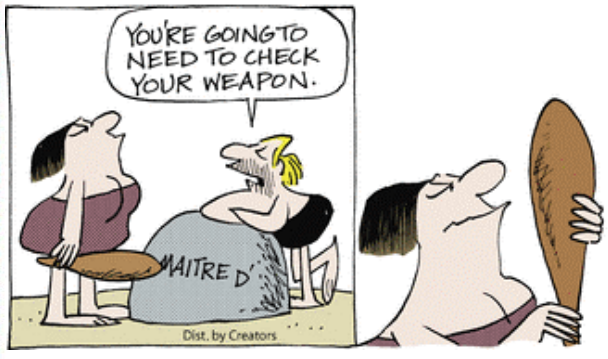
MA/CSSE 474

Theory of Computation

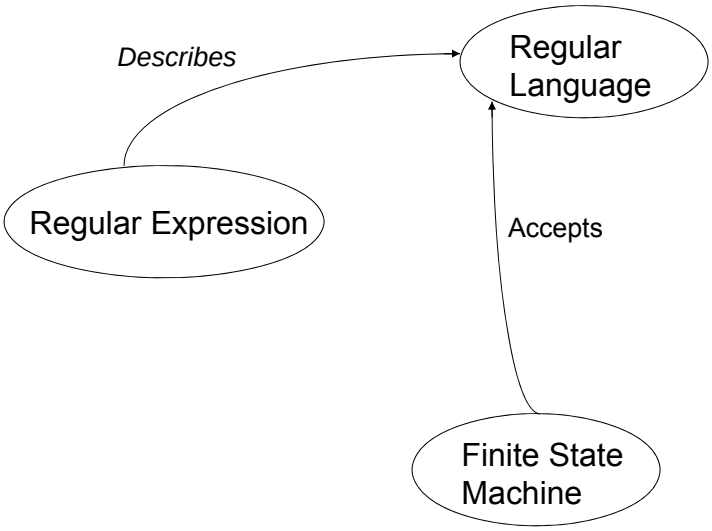
Regular Expressions Intro

Your Questions?

- Monday's class material
- Reading Assignments
- HW5 problems
- Anything else



Regular Languages



Regular Expressions

The regular expressions over an alphabet Σ are the strings that can be obtained from the following recursive definition:

1. $\varnothing$ is a regular expression.
2. ε is a regular expression.
3. Every element of Σ is a regular expression.
4. If α , β are regular expressions, then so is $\alpha\beta$.
5. If α , β are regular expressions, then so is $\alpha \cup \beta$.
6. If α is a regular expression, then so is α^*.
7. α is a regular expression, then so is α^+.
8. If α is a regular expression, then so is (α).
9. Nothing else is a regular expression.

.

Regular Expression Examples

If Σ = {a, b}, the following are regular expressions:

$\varnothing$

ε

a

$(a \cup b)^*$

$(abba \cup \varepsilon)^+ (a \cup bab)$

1. $\varnothing$ is a regular expression.
2. ε is a regular expression.
3. Every element of Σ is a regular expression.
4. If α , β are regular expressions, then so is $\alpha\beta$.
5. If α , β are regular expressions, then so is $\alpha \cup \beta$.
6. If α is a regular expression, then so is α^*.
7. α is a regular expression, then so is α^+.
8. If α is a regular expression, then so is (α).

Regular Expressions Define Languages

Define *L*, a **semantic interpretation function** for regular expressions (Let α and β be arbitrary regular expressions over alphabet Σ).

1. $L(\varnothing) = \varnothing$.
2. $L(\varepsilon) = \{\varepsilon\}$.
3. *If* $c \in \Sigma$, $L(c) = \{c\}$.
4. $L(\alpha\beta) = L(\alpha)\ L(\beta)$.
5. $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$.
6. $L(\alpha^*) = (L(\alpha))^*$.
7. $L(\alpha^+) = L(\alpha\alpha^*) = L(\alpha)\ (L(\alpha))^*$. If $L(\alpha)$ is equal to $\varnothing$, then $L(\alpha^+)$ is also equal to $\varnothing$. Otherwise $L(\alpha^+)$ is the language that is formed by concatenating together one or more strings drawn from $L(\alpha)$.
8. $L((\alpha)) = L(\alpha)$.

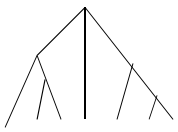
The Roles of the Rules

- Rules 1, 3, 4, 5, and 6 give the regular expression language its power to define sets.
- Rule 8 has grouping other expressions as its only role.
- Rules 2 and 7 appear to add functionality to the regular expression language, but they don't. They are very convenient, though.

1. $\varnothing$ is a regular expression.
2. ε is a regular expression.
3. Every element of Σ is a regular expression.
4. If α , β are regular expressions, then so is $\alpha\beta$.
5. If α , β are regular expressions, then so is $\alpha \cup \beta$.
6. If α is a regular expression, then so is α^*.
7. α is a regular expression, then so is α^+.
8. If α is a regular expression, then so is (α).

Operator Precedence in Regular Expressions

	Regular Expressions	**Arithmetic Expressions**
Highest	Kleene * and +	exponentiation
↕	concatenation	multiplication
Lowest	union	addition



a b* ∪ c d* $\qquad$ x y^2 + y x^2

Analyzing a Regular Expression

$L((a \cup b)^*b) = L((a \cup b)^*)\ L(b)$

$= (L(a \cup b))^*\ L(b)$

$= (L(a) \cup L(b))^*\ L(b)$

$= (\{a\} \cup \{b\})^*\ \{b\}$

$= \{a, b\}^*\ \{b\}.$

From English to reg exps

$L = \{w \in \{a, b\}^*: |w| \text{ is even}\}$

L = {*w* ∈ {0, 1}*: *w* is a binary representation of a positive multiple of 4}

$L = \{w \in \{a, b\}^*: w \text{ contains an odd number of a's}\}$

The Details Matter

$L(a^* \cup b^*) \neq L((a \cup b)^*)$

$L((ab)^*) \neq L(a^*b^*)$

More Regular Expression Examples

$(aa^*) \cup \varepsilon$ is equivalent to

$(a \cup \varepsilon)^*$ is equivalent to

$L = \{w \in \{a, b\}^*$: there is no more than one b in $w\}$

$L = \{w \in \{a, b\}^*$: no two consecutive letters in w are the same$\}$

The Details Matter

L_1 = {$w \in$ {a, b}* : every a is immediately followed by b}

A regular expression for L_1:

A FSM for L_1:

L_2 = {$w \in$ {a, b}* : every a has a matching b somewhere}

A regular expression for L_2:

A FSM for L_2:

Simplifying Regular Expressions

Regex's describe sets:

- Union is commutative: $\alpha \cup \beta = \beta \cup \alpha$.
- Union is associative: $(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma)$.
- $\varnothing$ is the identity for union: $\alpha \cup \varnothing = \varnothing \cup \alpha = \alpha$.
- Union is idempotent: $\alpha \cup \alpha = \alpha$.

Concatenation:

- Concatenation is associative: $(\alpha\beta)\gamma = \alpha(\beta\gamma)$.
- ε is the identity for concatenation: $\alpha\, \varepsilon = \varepsilon\, \alpha = \alpha$.
- $\varnothing$ is a zero for concatenation: $\alpha\, \varnothing = \varnothing\, \alpha = \varnothing$.

Concatenation distributes over union:

- $(\alpha \cup \beta)\, \gamma = (\alpha\, \gamma) \cup (\beta\, \gamma)$.
- $\gamma\, (\alpha \cup \beta) = (\gamma\, \alpha) \cup (\gamma\, \beta)$.

Kleene star:

- $\varnothing^* = \varepsilon$.
- $\varepsilon^* = \varepsilon$.
- $(\alpha^*)^* = \alpha^*$.
- $\alpha^*\alpha^* = \alpha^*$.
- $(\alpha \cup \beta)^* = (\alpha^*\beta^*)^*$.

Kleene's Theorem

Finite state machines and regular expressions define the same class of languages.

To prove this, we must show:

Theorem: Any language that can be defined by a regular expression can be accepted by some FSM and so is regular.
We do **reg. exp. → NDFSM** because it is easiest, and this is sufficient because we already know **NDFSM → DFSM**)

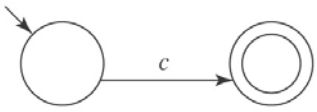
Theorem: Every regular language (i.e., every language that can be accepted by some DFSM) can be defined with a regular expression.

For Every Regular Expression There is a Corresponding FSM

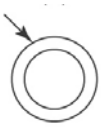
We'll show this by construction. An FSM for:

∅:

A single element c of Σ:

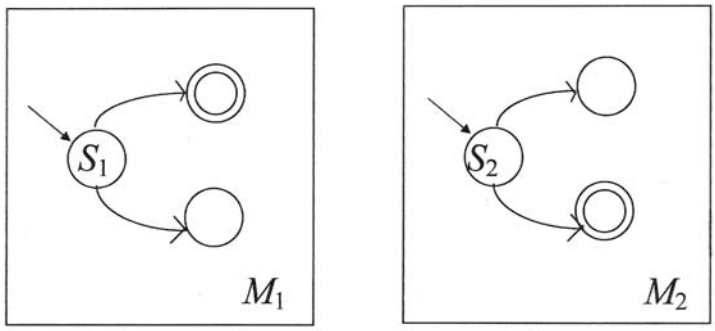


ε :



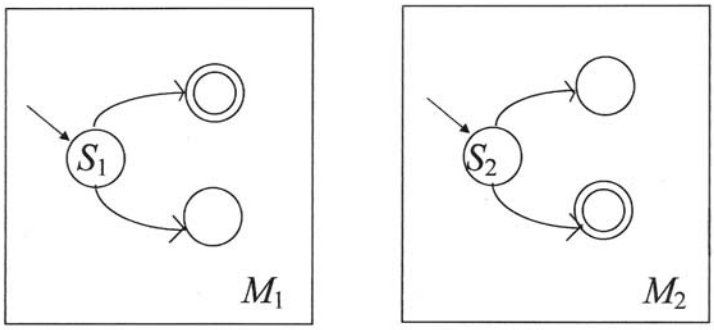
Union

If α is the regular expression $\beta \cup \gamma$ and if both $L(\beta)$ and $L(\gamma)$ are regular:



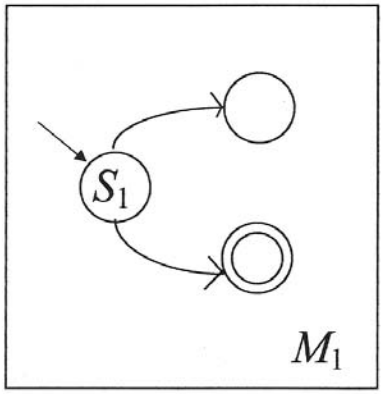
Concatenation

If α is the regular expression $\beta\gamma$ and if both $L(\beta)$ and $L(\gamma)$ are regular:



Kleene Star

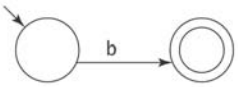
If α is the regular expression β^* and if $L(\beta)$ is regular:



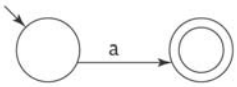
An Example

$(b \cup ab)^*$

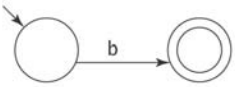
An FSM for b



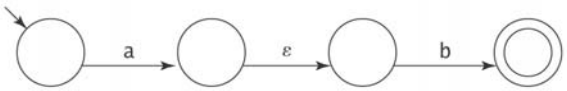
An FSM for a



An FSM for b



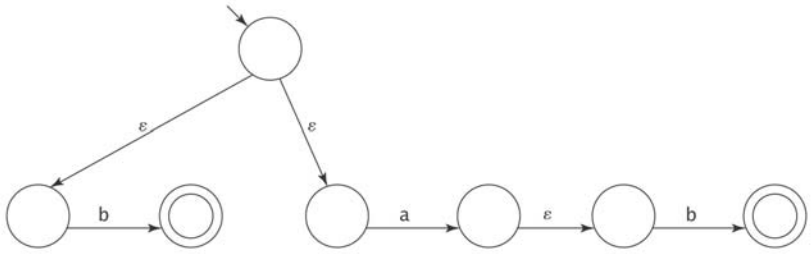
An FSM for ab:



An Example

(b ∪ ab)*

An FSM for (b ∪ ab):



An Example

(b ∪ ab)*

An FSM for (b ∪ ab)*: