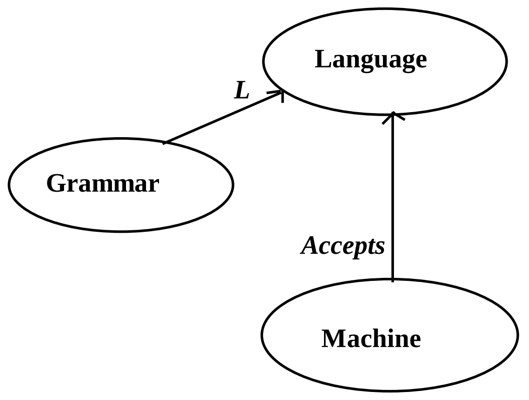
What Is a Language?

Do Homework 2.

Grammars, Languages, and Machines



Strings: the Building Blocks of Languages

An **alphabet** is a finite set of **symbols**:	English alphabet: {A, B, C, …, Z}
	Binary alphabet: {0, 1}

A **string** over an alphabet is a finite sequence of symbols drawn from the alphabet.

English string: happynewyear
binary string: 1001101

We will generally omit " " from strings unless doing so would lead to confusion.

The set of all possible strings over an alphabet Σ is written Σ*.
binary string: 1001101 ∈ {0,1}*

The shortest string contains no characters. It is called the **empty string** and is written " " or ε (epsilon).

The set of all possible strings over an alphabet Σ is written Σ*.

More on Strings

The **length** of a string is the number of symbols in it.
$|\varepsilon| = 0$
$|1001101| = 7$

A string a is a **substring** of a string b if a occurs contiguously as part of b.

aaa	is a substring of	aaabbbaaa
aaaaaa	is not a substring of	aaabbbaaa

Every string is a substring (although not a proper substring) of itself.

ε is a substring of every string. Alternatively, <u>we can match ε anywhere</u>.

Notice the analogy with sets here.

Operations on Strings

Concatenation: The **concatenation** of two strings x and y is written $x \parallel y$, $x \cdot y$, or xy and is the string formed by appending the string y to the string x.

$|xy| = |x| + |y|$

If $x = \varepsilon$ and $y =$ “food”, then xy =
If $x =$ “good” and $y =$ “bye”, then |xy| =

Note: $x \cdot \varepsilon = \varepsilon \cdot x = x$ for all strings x.

Replication: For each string w and each natural number i, the string w^i is defined recursively as

$w^0 = \varepsilon$
$w^i = w^{i-1} w$ for each $i \geq 1$

Like exponentiation, the replication operator has a high precedence.

Examples:
$a^3 =$
$(bye)^2 =$
$a^0b^3 =$

String Reversal

An inductive definition:

(1) If $|w| = 0$ then $w^R = w = \varepsilon$
(2) If $|w| \geq 1$ then $\exists\, a \in \Sigma$: $w = u \cdot a$
(a is the last character of w)
and
$w^R = a \cdot u^R$

Example:
$(abc)^R =$

More on String Reversal

Theorem: If w, x are strings, then $(w \cdot x)^R = x^R \cdot w^R$

Example: $(dogcat)^R = (cat)^R \cdot (dog)^R = tacgod$

Proof (by induction on |x|):

Basis: $|x| = 0$. Then $x = \varepsilon$, and $(w \cdot x)^R = (w \cdot \varepsilon)^R = (w)^R = \varepsilon \cdot w^R = \varepsilon^R \cdot w^R = x^R \cdot w^R$

Induction Hypothesis: If $|x| \leq n$, then $(w \cdot x)^R = x^R \cdot w^R$

Induction Step: Let $|x| = n + 1$. Then $x = u\, a$ for some character a and $|u| = n$

$$\begin{array}{lll}
(w \cdot x)^R & = (w \cdot (u \cdot a))^R & \\
 & = ((w \cdot u) \cdot a)^R & \text{associativity} \\
 & = a \cdot (w \cdot u)^R & \text{definition of reversal} \\
 & = a \cdot u^R \cdot w^R & \text{induction hypothesis} \\
 & = (u \cdot a)^R \cdot w^R & \text{definition of reversal} \\
 & = x^R \cdot w^R &
\end{array}$$

$$\underbrace{d\ o\ g}_{w}\ \underbrace{\underbrace{c\ a}_{u}\ \underbrace{t}_{a}}_{x}$$

Defining a Language

A **language** is a (finite or infinite) set of finite length strings over a finite alphabet Σ.

Example: Let Σ = {a, b}

Some languages over Σ: ∅, {ε}, {a, b}, {ε, a, aa, aaa, aaaa, aaaaa}

The language Σ* contains an infinite number of strings, including: ε, a, b, ab, ababaaa

Example Language Definitions

L = {x ∈ {a, b}* : all a's precede all b's}

L = {x : ∃y ∈ {a, b}* : x = ya}

L = {a^n, n ≥ 0 }

L = a^n (If we say nothing about the range of n, we will assume that it is drawn from N, i.e., n ≥ 0.)

L = {x#y: x,y ∈ {0-9}* and square(x) = y}

L = {} = ∅ (the empty language—not to be confused with {ε}, the language of the empty string)

Techniques for Defining Languages

Languages are sets. Recall that, for sets, it makes sense to talk about ***enumerations*** and ***decision procedures***. So, if we want to provide a computationally effective definition of a language we could specify either a

- Language **generator**, which enumerates (lists) the elements of the language, or a
- Language **recognizer**, which decides whether or not a candidate string is in the language and returns True if it is and False if it isn't.

Example: The logical definition: L = {x : ∃y ∈ {a, b}* : x = ya} can be turned into either a language generator or a language recognizer.

How Large are Languages?

- The smallest language over any alphabet is ∅. |∅| = 0
- The largest language over any alphabet is Σ*. |Σ*| = ?
  - If Σ = ∅ then Σ* = {ε} and |Σ*| = 1
  - If Σ ≠ ∅ then |Σ*| is countably infinite because its elements can be enumerated in 1 to 1 correspondence with the integers as follows:
    1. Enumerate all strings of length 0, then length 1, then length 2, and so forth.
    2. Within the strings of a given length, enumerate them lexicographically. E.g., aa, ab, ba, bb

- So all languages are either finite or countably infinite. Alternatively, all languages are *countable*.

Operations on Languages 1

Normal set operations: **union**, **intersection**, **difference**, **complement**…

Examples: Σ = {a, b} L_1 = strings with an even number of a's
L_2 = strings with no b's

$L_1 \cup L_2$ =
$L_1 \cap L_2$ =
$L_2 - L_1$ =
$\neg(L_2 - L_1)$ =

Operations on Languages 2

Concatenation: (based on the definition of concatenation of strings)

If L_1 and L_2 are languages over Σ, their concatenation $L = L_1 L_2$, sometimes $L_1 \cdot L_2$, is
$\{w \in \Sigma^*: w = x\, y \text{ for some } x \in L_1 \text{ and } y \in L_2\}$

Examples:

$L_1 = \{\text{cat, dog}\}$	$L_2 = \{\text{apple, pear}\}$	$L_1 L_2 = \{\text{catapple, catpear, dogapple, dogpear}\}$
$L_1 = \{a^n: n \geq 1\}$	$L_2 = \{a^n: n \leq 3\}$	$L_1 L_2 =$

Identities:

$L\varnothing = \varnothing L = \varnothing \quad \forall L$ (analogous to multiplication by 0)

$L\{\varepsilon\} = \{\varepsilon\}L = L \quad \forall L$ (analogous to multiplication by 1)

Replicated concatenation:

$L^n = L \cdot L \cdot L \cdot \ \ldots \cdot L$ (n times)

$L^1 = L$

$L^0 = \{\varepsilon\}$

Example:

$L = \{\text{dog, cat, fish}\}$

$L^0 = \{\varepsilon\}$

$L^1 = \{\text{dog, cat, fish}\}$

$L^2 = \{\text{dogdog, dogcat, dogfish, catdog, catcat, catfish, fishdog, fishcat, fishfish}\}$

Concatenating Languages Defined Using Variables

$L_1 = a^n = \{a^n : n \geq 0\}$ $\qquad$ $L_2 = b^n = \{b^n : n \geq 0\}$

$L_1 L_2 = \{a^n : n \geq 0\}\{b^n : n \geq 0\} = \{ a^n b^m : n,m \geq 0\}$ (common mistake:) $\neq a^n b^n = \{ a^n b^n : n \geq 0\}$

Note: The scope of any variable used in an expression that invokes replication will be taken to be the entire expression.

$L = 1^n 2^m$

$L = a^n b^m a^n$

Operations on Languages 3

Kleene Star (or Kleene closure): $\mathbf{L^*} = \{w \in \Sigma^* : w = w_1\, w_2 \ldots w_k \text{ for some } k \geq 0 \text{ and some } w_1, w_2, \ldots w_k \in L\}$

Alternative definition: $\mathbf{L^*} = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$

Note: $\forall L, \varepsilon \in L^*$

Example:

$L = \{\text{dog, cat, fish}\}$

$L^* = \{\varepsilon, \text{dog, cat, fish, dogdog, dogcat, fishcatfish, fishdogdogfishcat}, \ldots\}$

Another useful definition: $\mathbf{L^+} = L\, L^*$ $\qquad$ (L^+ is the **closure** of L under concatenation)

Alternatively, $\mathbf{L^+} = L^1 \cup L^2 \cup L^3 \cup \ldots$

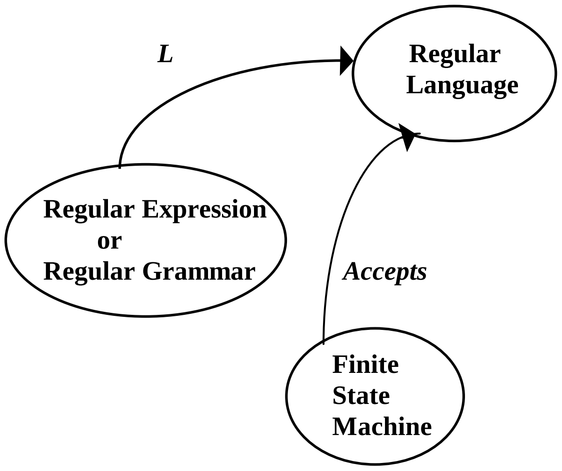
$L^+ = L^* - \{\varepsilon\}$ $\qquad$ if $\varepsilon \notin L$

$L^+ = L^*$ $\qquad$ if $\varepsilon \in L$

Regular Languages

Read Supplementary Materials: Regular Languages and Finite State Machines: Regular Languages
Do Homework 3.

Regular Grammars, Languages, and Machines



"Pure" Regular Expressions

The **regular expressions** over an alphabet Σ are all strings over the alphabet Σ ∪ {"(", ")", ∅, ∪, *} that can be obtained as follows:

1. ∅ and each member of Σ is a regular expression.
2. If α , β are regular expressions, then so is αβ
3. If α , β are regular expressions, then so is α∪β.
4. If α is a regular expression, then so is α*.
5. If α is a regular expression, then so is (α).
6. Nothing else is a regular expression.

If Σ = {a,b} then these are regular expressions: ∅, a, bab, a∪b , (a∪b)*a*b*

So far, regular expressions are just (finite) strings over some alphabet, Σ ∪ {"(", ")", ∅, ∪, *}.

Regular Expressions Define Languages

Regular expressions define languages via a **semantic interpretation function** we'll call L:
1. L(∅) = ∅ and L(a) = {a} for each a ∈ Σ
2. If α , β are regular expressions, then
L(αβ) = L(α)·L(β)
= all strings that can be formed by concatenating to some string from L(α) some string from L(β).
Note that if either α or β is ∅, then its language is ∅, so there is nothing to concatenate and the result is ∅.
3. If α , β are regular expressions, then L(α∪β) = L(α) ∪ L(β)
4. If α is a regular expression, then L(α*) = L(α)*
5. L((α)) = L(α)

A language is **regular** if and only if it can be described by a regular expression.

A regular expression is always finite, but it may describe a (countably) infinite language.

Regular Languages

An equivalent definition of the class of regular languages over an alphabet Σ:
The ***closure*** of the languages

$\{a\}\ \forall a \in \Sigma$ and $\varnothing$ [1]

with respect to the functions:

- concatenation, [2]
- union, and [3]
- Kleene star. [4]

In other words, the class of regular languages is the smallest set that includes all elements of [1] and that is closed under [2], [3], and [4].

"Closure" and "Closed"

Informally, a set can be defined in terms of a (usually small) starting set and a group of functions over elements from the set. The functions are applied to members of the set, and if anything new arises, it's added to the set. The resulting set is called the **closure** over the initial set and the functions. Note that the functions(s) may only be applied a **finite** number of times.

Examples:

The set of natural numbers **N** can be defined as the closure over {0} and the successor (succ(n) = n+1) function.
Regular languages can be defined as the closure of $\{a\}\ \forall a\in\Sigma$ and $\varnothing$ and the functions of concatenation, union, and Kleene star.

We say a set is **closed** over a function if applying the function to arbitrary elements in the set does not yield any new elements.

Examples:

The set of natural numbers **N** is closed under multiplication.
Regular languages are closed under intersection.

See *Supplementary Materials—Review of Mathematical Concepts* for more formal definitions of these terms.

Examples of Regular Languages

$L(\ a^*b^*\ ) =$
$L(\ (a \cup b)\ ) =$
$L(\ (a \cup b)^*\ ) =$
$L(\ (a\cup b)^*a^*b^*) =$
$L = \{w \in \{a,b\}^* : |w| \text{ is even}\}$
$L = \{w \in \{a,b\}^* : w \text{ contains an odd number of a's}\}$

Augmenting Our Notation

It would be really useful to be able to write ε in a regular expression.

Example: $(a \cup \varepsilon)\ b$ (Optional a followed by b)

But we'd also like a minimal definition of what constitutes a regular expression. Why?

Observe that

$\varnothing^0 = \{\varepsilon\}$ (since 0 occurrences of the elements of any set generates the empty string), so
$\varnothing^* = \{\varepsilon\}$

So, without changing the set of languages that can be defined, we can add ε to our notation for regular expressions if we specify that

$L(\varepsilon) = \{\varepsilon\}$

We're essentially treating ε the same way that we treat the characters in the alphabet.
Having done this, you'll probably find that you rarely need $\varnothing$ in any regular expression.

More Regular Expression Examples

$L(\ (aa^*) \cup \varepsilon\ ) =$
$L(\ (a \cup \varepsilon)^*\ ) =$
$L = \{ w \in \{a,b\}^* :$ there is no more than one b$\}$
$L = \{ w \in \{a,b\}^* :$ no two consecutive letters are the same$\}$

Further Notational Extensions of Regular Expressions

- A fixed number of concatenations: α^n means $\alpha\alpha\alpha\alpha\ldots\alpha$ (n times).
- At Least 1: α^+ means 1 or more occurrences of α concatenated together.
- Shorthands for denoting sets, such as ranges, e.g., (A-Z) or (letter-letter)
  Example: $L = (A\text{-}Z)^+((A\text{-}Z)\cup(0\text{-}9))^*$

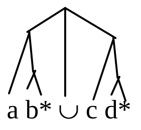
- A replicated regular expression α^n, where n is a constant.
  Example: $L = (0 \cup 1)^{20}$

- Intersection: $\alpha\cap\beta$ (we'll prove later that regular languages are closed under intersection)
  Example: $L = (a^3)^* \cap (a^5)^*$

Operator Precedence in Regular Expressions

Regular expressions are strings in the language of regular expressions. Thus to interpret them we need to:
1. Parse the string
2. Assign a meaning to the parse tree

Parsing regular expressions is a lot like parsing arithmetic expressions. To do it, we must assign precedence to the operators:

		Regular Expressions	**Arithmetic Expressions**
Highest		Kleene star	exponentiation
	↕	concatenation	multiplication
		intersection	
Lowest		union	addition
			$x\ y^2 + i\ j^2$

Regular Expressions and Grammars

Recall that grammars are language generators. A grammar is a recipe for creating strings in a language.

Regular expressions are analogous to grammars, but with two special properties:

1. The have limited power. They can be used to define only regular languages.
2. They don't look much like other kinds of grammars, which generally are composed of sets of production rules.

But we can write more "standard" grammars to define exactly the same languages that regular expressions can define. Specifically, any such grammar must be composed of rules that:

- have a left hand side that is a single nonterminal
- have a right hand side that is ε, or a single terminal, or a single terminal followed by a single nonterminal.

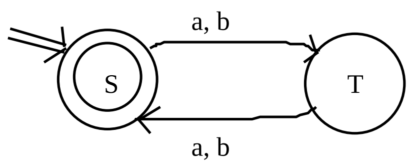
Regular Grammar Example

L={w ∈ {a, b}* : |w| is even}

((aa) ∪ (ab) ∪ (ba) ∪ (bb))*

Notice how these rules correspond naturally to a FSM:

$S \rightarrow \varepsilon$
$S \rightarrow aT$
$S \rightarrow bT$
$T \rightarrow a$
$T \rightarrow b$
$T \rightarrow aS$
$T \rightarrow bS$



Generators and Recognizers

$$\begin{tikzcd}
\text{Generator} \arrow[r] & \text{Language} & \text{Recognizer} \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
\begin{array}{c}\text{Regular Expressions}\\ \text{Regular Grammars}\end{array} \arrow[r] & \text{Regular Languages} & ? \arrow[l]
\end{tikzcd}$$