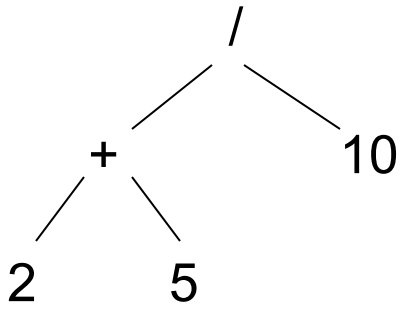
Languages and Strings

Chapter 2

Let's Look at Some Problems

```
int alpha, beta;
alpha = 3;
beta = (2 + 5) / 10;
```

(1) **Lexical analysis**: Scan the program and break it up into variable names, numbers, etc.
(2) **Parsing**: Create a tree that corresponds to the sequence of operations that should be executed, e.g.,



(3) **Optimization**: Realize that we can skip the first assignment since the value is never used and that we can precompute the arithmetic expression, since it contains only constants.
(4) **Termination**: Decide whether the program is guaranteed to halt.
(5) **Interpretation**: Figure out what (if anything) useful it does.

A Framework for Analyzing Problems

We need a single framework in which we can analyze a very diverse set of problems.

The framework we will use is

Language Recognition

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

Strings

A ***string*** is a finite sequence, possibly empty, of symbols drawn from some alphabet Σ.

- ε is the empty string.
- Σ^* is the set of all possible strings over an alphabet Σ.

Alphabet name	***Alphabet symbols***	***Example strings***
The English alphabet	{a, b, c, …, z}	ε, aabbcg, aaaaa
The binary alphabet	{0, 1}	ε, 0, 001100
A star alphabet	{★ , ✪ , ✯ , ✶, ✡, ✰}	ε, ✪✪, ✪✶✶✯★✯
A music alphabet	{w, h, q, e, x, r, ●}	ε, w l h h l hqq l

Functions on Strings

Counting: $|s|$ is the number of symbols in s.

$|\varepsilon| = 0$
$|\texttt{1001101}| = 7$

$\#_c(s)$ is the number of times that c occurs in s.

$\#_a(\texttt{abbaaa}) = 4$.

More Functions on Strings

Concatenation: *st* is the ***concatenation*** of *s* and *t*.

If x = `good` and y = `bye`, then xy = `goodbye`.

Note that $|xy| = |x| + |y|$.

ε in string concatenation:

$\forall x\ (x\,\varepsilon = \varepsilon\,x = x)$.

Concatenation is associative:

$\forall s, t, w\ ((st)w = s(tw))$.

More Functions on Strings

Replication: For each string *w* and each natural number *i*, the string w^i is:

$w^0 = \varepsilon$
$w^{i+1} = w^i\, w$

Examples:

$\mathtt{a}^3 = \mathtt{aaa}$
$(\mathtt{bye})^2 = \mathtt{byebye}$
$\mathtt{a}^0\mathtt{b}^3 = \mathtt{bbb}$

More Functions on Strings

Reverse: For each string w, w^R is defined as:

if $|w| = 0$ then $w^R = w = \varepsilon$

if $|w| \geq 1$ then:

$\exists a \in \Sigma\ (\exists u \in \Sigma^*\ (w = ua))$.

So define $w^R = a\ u^R$.

Concatenation and Reverse of Strings

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

Example:

$(\texttt{nametag})^R = (\texttt{tag})^R\,(\texttt{name})^R = \texttt{gateman}$

Concatenation and Reverse of Strings

Proof: By induction on $|x|$:

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

$\forall n \geq 0\ (((|x| = n) \rightarrow ((w\ x)^R = x^R\ w^R)) \rightarrow$
$((|x| = n + 1) \rightarrow ((w\ x)^R = x^R\ w^R)))$:

Consider any string x, where $|x| = n + 1$. Then $x = u\ a$ for some character a and $|u| = n$. So:

$(w\ x)^R$	$= (w\ (u\ a))^R$	rewrite x as ua
	$= ((w\ u)\ a)^R$	associativity of concatenation
	$= a\ (w\ u)^R$	definition of reversal
	$= a\ (u^R\ w^R)$	induction hypothesis
	$= (a\ u^R)\ w^R$	associativity of concatenation
	$= (ua)^R\ w^R$	definition of reversal
	$= x^R\ w^R$	rewrite ua as x

Relations on Strings

`aaa`	is a ***substring*** of	`aaabbbaaa`
`aaaaaa`	is not a substring of	`aaabbbaaa`
`aaa`	is a ***proper substring*** of	`aaabbbaaa`

Every string is a substring of itself.

ε is a substring of every string.

The Prefix Relations

s is a ***prefix*** of *t* iff: $\exists x \in \Sigma^* (t = sx)$.

s is a ***proper prefix*** of *t* iff: *s* is a prefix of *t* and $s \neq t$.

Examples:

The ***prefixes*** of `abba` are: ε, `a`, `ab`, `abb`, `abba`.
The ***proper prefixes*** of `abba` are: ε, `a`, `ab`, `abb`.

Every string is a prefix of itself.

ε is a prefix of every string.

The Suffix Relations

s is a ***suffix*** of *t* iff: $\exists x \in \Sigma^* (t = xs)$.

s is a ***proper suffix*** of *t* iff: *s* is a suffix of *t* and $s \neq t$.

Examples:

The ***suffixes*** of `abba` are: ε, `a`, `ba`, `bba`, `abba`.
The ***proper suffixes*** of `abba` are: ε, `a`, `ba`, `bba`.

Every string is a suffix of itself.

ε is a suffix of every string.

Defining a Language

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

Examples: Let Σ = {a, b}

Some languages over Σ:

$\varnothing$,
{ε},
{a, b},
{ε, a, aa, aaa, aaaa, aaaaa}

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

Example Language Definitions

$L = \{x \in \{\texttt{a}, \texttt{b}\}^* : \text{all } \texttt{a}\text{'s precede all } \texttt{b}\text{'s}\}$

ε, `a`, `aa`, `aabbb`, and `bb` are in L.

`aba`, `ba`, and `abc` are not in L.

Example Language Definitions

$L = \{x : \exists y \in \{\mathtt{a}, \mathtt{b}\}^* : x = y\mathtt{a}\}$

Simple English description:

More Example Language Definitions

$L = \{\} = \varnothing$

$L = \{\varepsilon\}$

English not Well-Defined

L = {w: w is a sentence in English}.

Examples:

Kerry hit the ball.

Colorless green ideas sleep furiously.

The window needs fixed.

Ball the Stacy hit blue.

A Halting Problem Language

L = {w: w is a C program that halts on all inputs}.

- Well specified
- Useful

Enumeration

- More useful: ***lexicographic order***
  - Shortest first
  - Within a length, dictionary order
  - This way, can enumerate a language

The lexicographic enumeration of:

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

How Large is a Language?

The smallest language over any Σ is $\varnothing$, with cardinality 0.

The largest is Σ^*. How big is it?

How Large is a Language?

Theorem: If $\Sigma \neq \emptyset$ then Σ^* is countably infinite.

Proof: The elements of Σ^* can be lexicographically enumerated by the following procedure:

- Enumerate all strings of length 0, then length 1, then length 2, and so forth.
- Within the strings of a given length, enumerate them in dictionary order.

This enumeration is infinite since there is no longest string in Σ^*. Since there exists an infinite enumeration of Σ^*, it is countably infinite.

■

How Large is a Language?

So the smallest language has cardinality 0.

The largest is countably infinite.

So every language is either finite or countably infinite.

How Many Languages Are There?

Theorem: If $\Sigma \neq \varnothing$ then the set of languages over Σ is uncountably infinite.

Proof: The set of languages defined on Σ is $\mathcal{P}(\Sigma^*)$. Σ^* is countably infinite. If S is a countably infinite set, $\mathcal{P}(S)$ is uncountably infinite. So $\mathcal{P}(\Sigma^*)$ is uncountably infinite. ■

Functions on Languages

- Set operations
  - Union
  - Intersection
  - Complement
- Language operations
  - Concatenation
  - Kleene star

Concatenation of Languages

If L_1 and L_2 are languages over Σ:

$$L_1L_2 = \{w \in \Sigma^* : \exists s \in L_1 \, (\exists t \in L_2 \, (w = st))\}$$

Examples:

L_1 = {`cat`, `dog`}
L_2 = {`apple`, `pear`}
L_1 L_2 ={`catapple`, `catpear`, `dogapple`, `dogpear`}

$L\{\varepsilon\} = \{\varepsilon\}L = L$

$L \varnothing = \varnothing L = \varnothing$

Kleene Star

- **Kleene operator**

$$L^* = \{\varepsilon\} \cup \{w \in \Sigma^* : \exists k \geq 1 \; (\exists w_1, w_2, \ldots w_k \in L \;\; (w = w_1 \, w_2 \ldots w_k))\}$$

- In other words, L^* is the set of strings that can be formed by concatenating together zero or more strings in L
- Is **closed under concatenation**

Example: L = {`dog`, `cat`, `fish`}

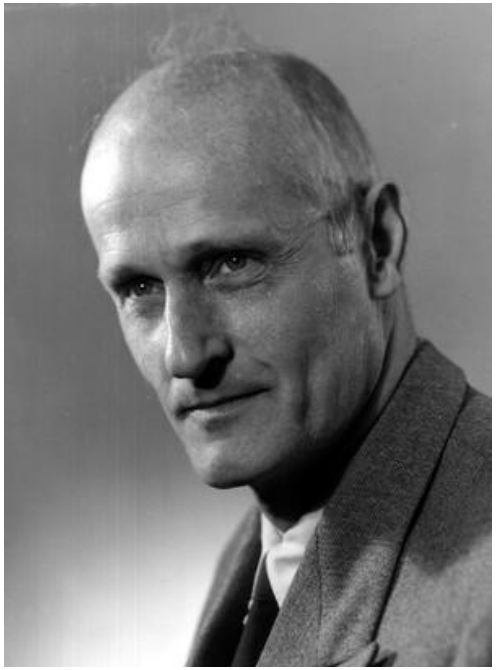
L^* = {ε, `dog`, `cat`, `fish`, `dogdog`, `dogcat`, `fishcatfish`, `fishdogdogfishcat`, …}

If $L = \varnothing$, L^* = ?

If $L = \{\varepsilon\}$, L^* = ?

Is L^* the concatenation closure of L?

Stephen Cole Kleene



- 1909 – 1994, mathematical logician
- One of many distinguished students (e.g., Alan Turing) of Alonzo Church at Princeton.
- Best known as a founder of the branch of mathematical logic known as recursion theory.
- Also invented regular expressions.

- Kleeneness is next to Godelness
  - Cleanliness is next to Godliness

- Obituaries from New York Times
  - Modern computer science, in large part, grew out of recursion theory, and his work has been tremendously influential for years. One of the reasons for the importance of recursion theory is that it gives us a way of showing that some mathematical problems can never be solved, no matter how much computing power is available
  - https://www.nytimes.com/1994/01/27/obituaries/stephen-c-kleene-is-dead-at-85-was-leader-in-computer-science.html

A photo by me

The $^+$ Operator

Sometimes useful to require at least one element of L be selected

$L^+ = L\ L^*$

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

L^+ is the **closure** of L under concatenation.

- recall closure of L needs to contain L

Concatenation and Reverse of Languages

Theorem: $(L_1 L_2)^R = L_2^R L_1^R$.

Proof:

$\forall x (\forall y ((xy)^R = y^R x^R))$ Theorem 2.1

$(L_1 L_2)^R = \{(xy)^R : x \in L_1 \text{ and } y \in L_2\}$ Definition of concatenation of languages

$= \{y^R x^R : x \in L_1 \text{ and } y \in L_2\}$ Lines 1 and 2

$= L_2^R L_1^R$ Definition of concatenation of languages

What About Meaning?

$A^nB^n = \{a^n b^n : n \geq 0\}$.

Do these strings mean anything?

But some languages are useful because their strings have meanings.

- English, Chinese, Java, C++, Perl

Need a precise way to map each string to its meaning (semantics).

Semantic Interpretation Functions

A semantic interpretation function assigns meanings to the strings of a language.

• Language is mostly infinite, no way to map each string
• Must define a function that knows the meanings of basic units and can compose those meanings according to some fixed set of rules, to build meanings for larger expressions

Compositional semantic interpretation function

How about English? Compositional?

- I gave him a mug
- I'm going to give him a piece of my mind

Semantic Interpretation Functions

• When we define a formal language for a specific purpose, we design it so that there exists a compositional semantic interpretation function

  • For example, C++ and Java

• One significant property of semantic interpretation functions for useful languages is that they are generally not one-to-one

  • “Chocolate, please”, “I’ d like chocolate”, “I’ ll have chocolate” …

  • “x++”, “x = x + 1”