(NB. Pages 21 - 39 are intended for those who need repeated study in formal languages)

Formal languages

Basic concepts for symbols, strings and languages:

Alphabet

A finite set of symbols.

Example:

Σ_b = { 0,1 } binary alphabet
Σ_s = { A,B,C,...,Z,Å,Ä,Ö } Swedish characters
Σ_r = { WHILE,IF,BEGIN,... } reserved words

String

A finite sequence of symbols from an alphabet.

Example:

10011 from Σ_b
KALLE from Σ_s
WHILE DO BEGIN from Σ_r

Length of a string

Number of symbols in the string.

Example:

x arbitrary string, $|x|$ length of the string x
$|10011| = 5$ according to Σ_b
$|WHILE| = 5$ according to Σ_s
$|WHILE| = 1$ according to Σ_r

Empty string

The empty string is denoted ε, $|\varepsilon| = 0$

Concatenation

Two strings x and y are joined together $x \bullet y = xy$

Example:

$x = AB$, $y = CDE$ produce $x \bullet y = ABCDE$
$|xy| = |x| + |y|$
$xy \neq yx$ (not commutative)
$\varepsilon x = x\varepsilon = x$

String exponentiation

$x^0 = \varepsilon$
$x^1 = x$
$x^2 = xx$
$x^n = x \bullet x^{n-1},\ n >= 1$

Substrings: Prefix, suffix.

Example: x = abc

Prefix: Substring at the beginning.

Prefix of x: abc (improper as the prefix = x), ab, a, ε

Suffix: Substring at the end.

Suffix of x: abc (improper as the suffix = x), bc, c, ε

Languages

A finite or infinite set of strings which can be constructed from a special alphabet.

Alternatively: a subset of all the strings which can be constructed from an alphabet.

$\varnothing$ = the empty language. NB! $\{\varepsilon\} \neq \varnothing$.

Example: $\Sigma = \{0,1\}$

$L_1 = \{00,01,10,11\}$ all strings of length 2

$L_2 = \{1,01,11,001,...,111, ...\}$
all strings which finish on 1

$L_3 = \varnothing$ all strings of length 1 which finish on 01

Σ^* denotes the set of all strings which can be constructed from the alphabet.

* = closure, Kleene closure.
+ = positive closure.

e.g. $\Sigma = \{0,1\}$
$\Sigma^* = \{\varepsilon, 0,1,00,01,...,111,101,...\}$
$\Sigma^+ = \Sigma^* - \{\varepsilon\} = \{0,1,00,01,...\}$

Operations on languages

Concatenation

L, M are languages.

$L \bullet M = LM = \{xy | x \in L \text{ and } y \in M\}$

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

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

Example: $L = \{ab,cd\}$ $M = \{uv,yz\}$
gives us
$LM = \{abuv,abyz,cduv,cdyz\}$

Exponents of languages

$L^0 = \{\varepsilon\}$
$L^1 = L$
$L^2 = L \bullet L$
$L^n = L \bullet L^{n-1},\ n >= 1$

Union of languages

L, M are languages.

$L \cup M = \{x \mid x \in L \text{ or } x \in M\}$

Example: $L = \{ab, cd\}$, $M = \{uv, yz\}$

gives us $L \cup M = \{ab, cd, uv, yz\}$

Closure

$L^* = L^0 \cup L^1 \cup \ldots \cup L^\infty$

Positive closure

$L^+ = L^1 \cup L^2 \cup \ldots \cup L^\infty\ LL^* = L^* - \{\varepsilon\}$, if $\varepsilon \notin L$

$L^* = \{\varepsilon\} \cup L^+$

Example: $A = \{a, b\}$

$A^* = \{\varepsilon, a, b, aa, ab, ba, bb, \ldots\}$

= All possible sequences of a and b.

A language over A is always a subset of A*.

Regular expressions

Are used to describe simple languages, e.g. basic symbols, tokens.

Example: identifier = letter • (letter | digit)*

Regular expressions over an alphabet Σ denote a language (regular set).

Rules for constructing regular expressions

Σ is an alphabet, the regular expression r describes the language L_r, the regular expression s corresponds to the language L_s, etc.

Regular expression r	Language L_r
ε	$\{\varepsilon\}$
$a,\ a \in \Sigma$	$\{ a \}$ †
union: (s) | (t)	$L_s \cup L_t$
concatenation: (s).(t)	$L_s . L_t$
repetition: $(s)^*$	$L_s{}^*$
repetition: $(s)^+$	$L_s{}^+$

† Each symbol in the alphabet Σ is a regular expression which denotes {a}.

* = repetition, zero or more times.

\+ = repetition, one or more times.

Priorities	
Highest	*, +
	.
Lowest	|

Example: $\Sigma = \{a, b\}$

1. $r = a$ $L_r = \{a\}$
2. $r = a^*$ $L_r = \{\varepsilon, a, aa, aaa, \ldots\} = \{a\}^*$
3. $r = a|b$ $L_r = \{a, b\} = \{a\} \cup \{b\}$
4. $r = (a|b)^*$
   $L_r = \{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$
5. $r = (a^* b^*)^*$
   $L_r = \{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$
6. $r = a|ba^*$
   $L_r = \{a, b, ba, baa, baaa, \ldots\} = \{a \text{ or } ba^i \mid i \geq 0\}$

NB! $\{a^n b^n \mid n >= 0\}$ can not be described with regular expressions.

$r = a^* b^*$ gives us $L_r = \{a^i b^j \mid i, j >= 0\}$ does not work.

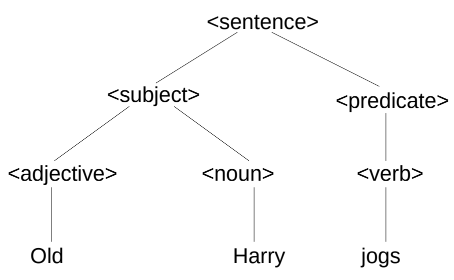
$r = (ab)^*$ gives us $L_r = \{(ab)^i \mid i >= 0\}$
$= \{\varepsilon, ab, abab, \ldots\}$ does not work.

Regular expressions can not "count" (have no memory).

Context-free grammars

Example: an English sentence

Sentence:	Old	Harry	jogs
Constituent:	subject		predicate
Word class:	adjective	noun	verb



A grammar is used to describe the syntax.
BNF (Backus-Naur form) 1960 (metalanguage to describe languages):

<sentence> → <subject> <predicate>

<subject> → <adjective> <noun>

<predicate> → <verb>

<adjective> → old | big | strong | ...

<noun> → Harry | brother | ...

<verb> → jogs | snores | sleeps | ...

- <sentence> is a *start symbol.*
- Symbols to the left of " $\rightarrow$ " are called *nonterminals*.
- Symbols not surrounded by "< >" are *terminals.*
- Each line is a *production.*

Symbol	Meaning
< ... >	syntactic classes
$\rightarrow$	"consists of", "is" (also "::=")
|	"or"

The grammar can be used to produce or derive sentences.

Example:<sentence> $\stackrel{*}{\Rightarrow}$ Old Harry jogs

where <sentence> is the start symbol and "$\stackrel{*}{\Rightarrow}$" means derivation in zero or more steps.

e.g. Derivation

<sentence> $\Rightarrow$ <subject> <predicate>
$\Rightarrow$ <adjective> <noun> <predicate>
$\Rightarrow$ Old <noun> <predicate>
$\Rightarrow$ Old Harry <predicate>
$\Rightarrow$ Old Harry <verb>
$\Rightarrow$ Old Harry jogs

Definition:

A CFG (Context-free grammar) is a quadruple (4 parts):

G = < N, Σ, P, S > where
N : nonterminals.
Σ : terminal symbols.
P : rules, productions of the form
$A \rightarrow a$ where $A \in N$ and $a \in (N \cup \Sigma)^*$
S : the start symbol, a nonterminal, $S \in N$.

(Sometimes $V = N \cup \Sigma$ is used, called the *vocabulary.*)

Example:

1.<number> $\rightarrow$ <no>
2.<no> $\rightarrow$ <no> <digit>
3. | <digit>
4.<digit> $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

N = { <number>, <no>, <digit> }
Σ = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
S = <number>

A grammar G denotes the language L(G),

L(G) is generated by the grammar G.

L(<number>) = { w | $w \in$ digit$^+$ }

Conventions	
$\alpha, \beta, \gamma \in V^*$	string of terminals and nonterminals
$A, B, C \in N$	nonterminals
$a, b, c \in \Sigma$	terminal symbols
$u, v, w, x, y, z \in \Sigma^*$	string of terminals

Derivation

$\alpha \Rightarrow \beta$ (pronounced "α derives β")

Formally: $\gamma A \theta \Rightarrow \gamma \delta \theta$ if we have $A \rightarrow \delta$

Example: <number> $\underset{rm}{\Rightarrow}$<no> $\underset{rm}{\Rightarrow}$ <no> <digit> $\underset{rm}{\Rightarrow}$
<no> 2 $\underset{rm}{\Rightarrow}$ <digit> 2 $\underset{rm}{\Rightarrow}$ 12

- <number> $\Rightarrow$ <no> direct derivation.
- <number> $\stackrel{*}{\Rightarrow}$ 12 several derivations (zero or more).
- <number> $\stackrel{+}{\Rightarrow}$ 12 several derivations (one or more).

Given G = < N, Σ, P, S > the language generated by G can be defined as L(G):

L(G) = { w | S $\stackrel{+}{\Rightarrow}$ w och $w \in \Sigma^*$ }

Sentential form

A string α is a *sentential form* in G if

S $\stackrel{*}{\Rightarrow}$ α and $\alpha \in V^*$ (string of terminals and nonterminals.)

Example: <u><no> <digit></u> is a sentential form in G(<number>).

Sentence

w is a *sentence* in G if S $\stackrel{+}{\Rightarrow}$ w and $w \in \Sigma^*$.

Example: <u>12</u> is a sentence in G(<number>).

Left derivation

$\underset{rm}{\Rightarrow}$ means that we replace the *leftmost* nonterminal by some appropriate right side.

Left sentential form

A sentential form which is part of a leftmost derivation.

Right derivation (canonical derivation)

$\underset{rm}{\Rightarrow}$ means that we replace the *rightmost* nonterminal by some appropriate right side.

Right sentential form

A sentential form which is part of a rightmost derivation.

Reverse rightmost derivation

12 $\underset{rm}{\Leftarrow}$ <digit> 2 $\underset{rm}{\Leftarrow}$ <no> 2 $\underset{rm}{\Leftarrow}$ <no> <digit>
$\underset{rm}{\Leftarrow}$ <no> $\underset{rm}{\Leftarrow}$ <number>

Handles

Consist of two parts:

1. A production $A \rightarrow \beta$
2. A position

If $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta w$, the production is $A \rightarrow \beta$ and the position after a is a handle of $\alpha\beta w$.

Example: The handle of <u><no> 2</u> is the production <digit> $\rightarrow$ 2 and the position after <no> because:

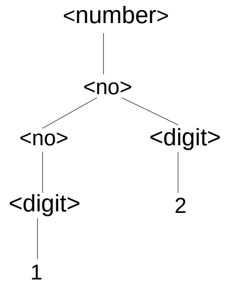
<number> $\underset{rm}{\Rightarrow}$ <no> $\underset{rm}{\Rightarrow}$ <no> <digit> $\underset{rm}{\Rightarrow}$ <no> 2 $\underset{rm}{\Rightarrow}$
<digit> 2 $\underset{rm}{\Rightarrow}$ 12

Informally: a handle is what we *reduce* to what and where to get the previous sentential form in a rightmost derivation.

Reduction

In reverse right derivation, find a right side in some rule according to the grammar in the given right sentential form and replace it with the corresponding left side, i.e. nonterminal.

Parse trees (derivation trees)



Parse tree för 12

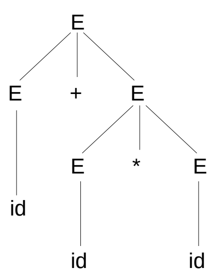
A parse tree can correspond to several different derivations.

Ambiguous grammars

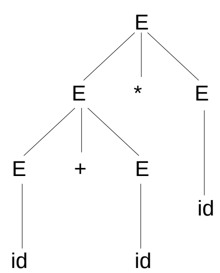
A grammar G is *ambiguous* if a sentence in G has several different parse trees.

e.g. $E \rightarrow E + E$
| E * E
| $E \uparrow E$
| id

<u>id+id*id</u> has two different parse trees:



id+(id*id)



id*(id+id)

Rewrite the grammar to make it unambiguous:

- +, * are to have the right priority and
- +, * are to be left associative while
- $\uparrow$ is to be right associative.

Example: a+b+c+d is interpreted as (a+b)+c+d.

$E \rightarrow E + T$ (left associative)
| T
$T \rightarrow T * F$ (left associative)
| F
$F \rightarrow P \uparrow F$ (right associative)
| P
$P \rightarrow$ id

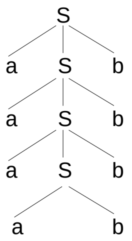
Home assignment:

Draw the parse trees for:

1. id * id * id
2. id $\uparrow$ id $\uparrow$ id

Example: The following grammar generates
$\{ a^n b^n \mid n >= 1 \}$.

$S \rightarrow a\,S\,b$
$\quad\mid a\,b$



e.g. $S \rightarrow 0\,S\,0$
$\mid$ 1 S 1
$\mid$ 0
$\mid$ 1

describes binary palindromes of odd length.

Home assignment:
Write a CFG for binary palindromes of all lengths >= 1,
i.e. ε is not included.

Excerpt from a Pascal grammar:

```
<goal> → <progdecl> .
<progdecl> → <prog_hedr> ; <block>
<prog_hedr> → program <idname> ( <idname_list> )
            | program <idname>
<block> → <decls> begin <stat_list> end
<decls> → <labels> <consts> <types> <vars> <procs>
<labels> → label <label_decl> ;
        | ε
<label_decl> → <label_decl> , <labelid>
            | <labelid>
<labelid> → <int>
          | <id>
<consts> → const <const_decls>
         | ε
<const_decls> → <const_decls> <const_decl_c>
              | <const_decl_c>
<const_decl_c> → <const_decl> ;
<const_decl> → <idname> = <const>
<types> → type <type_decls>
        | ε
<type_decls> → <type_decls> <type_decl_c>
            | <type_decl_c>
<type_decl_c> → <type_decl> ;
<type_decl> → <idname> = <type>
<vars> → var <var_decls>
       | ε
<var_decls> → <var_decls> <var_decl_c>
            | <var_decl_c>
<var_decl_c> → <var_decl> ;
<var_decl> → <id_list> : <type>
<procs> → <proc_decls>
        | ε
<proc_decls> → <proc_decls> <proc>
            | <proc>
```



```
<proc> → procedure <phead_c> forward ;
       | procedure <phead_c> <block> ;
       | function <fhead_c> forward ;
       | function <fhead_c> <block> ;
<fhead_c> → <fhead> ;
<fhead> → <idname> <params> : <type_id>
<phead_c> → <phead> ;
<phead> → <idname> <params>
        | ε
<params> → ( <param_list> )
        | ε
<param> → var <par_decl>
        | <par_decl>
        | ε
<par_decl> → <id_list> : <type_id>
<param_list> → <param_list> ; <param>
            | <param>
<id_list> → <id_list> , <id>
          | <id>
...
```