TDDD55 Compilers and interpreters

TDDE66 Compiler Construction

Formal Languages Part 1
Including Regular Expressions

IDA, Linköpings universitet, 2024.

Basic Concepts for Symbols, Strings, and Languages

- **Alphabet**
  A finite set of symbols.
- Example:
  $\Sigma_b = \{ 0,1 \}$ binary alphabet
  $\Sigma_s = \{ A,B,C,...,Z,Å,Ä,Ö \}$ Swedish characters
  $\Sigma_r = \{ WHILE,IF,BEGIN,... \}$ 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

Properties of Strings in Formal Languages
String Length, Empty String

LINKÖPINGS UNIVERSITET

- **Length of a string**
  - Number of symbols in the string.
- **Example:**
  - x arbitrary string, |x| length of the string x
  - |10011| = 5 according to $\sum_b$
  - |WHILE| = 5 according to $\sum_s$
  - |WHILE| = 1 according to $\sum_r$
- **Empty string**
  - The empty string is denoted ϵ, |ϵ| = 0

Properties of Strings in Formal Languages Concatenation, Exponentiation

LINKÖPINGS UNIVERSITET

- **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)
  - $\epsilon x = x \epsilon = x$
- **String exponentiation**
  - $x^0 = \epsilon$
  - $x^1 = x$
  - $x^2 = xx$
  - $x^n = x \bullet x^{n-1}$, $n >= 1$

Substrings: Prefix, Suffix

- Example:
  - x = abc

- Prefix: Substring at the beginning.
  - Prefix of x: abc (improper as the prefix equals x), ab, a, ϵ

- Suffix: Substring at the end.
  - Suffix of x: abc (improper as the suffix equals x), bc, c, ϵ

Languages

- A Language = 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.
  - $\emptyset$ = the empty language. NB! $\{\epsilon\} \neq \emptyset$.
- Example: S = {0,1}
  - L1 = {00,01,10,11} all strings of length 2
  - L2 = {1,01,11,001,...,111, ...} all strings which finish on 1
  - L3 = $\emptyset$ all strings of length 1 which finish on 01

Closure

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

- Closure types:
  - * = closure, Kleene closure
  - \+ = positive closure

- Example: S = {0,1}
  - $\Sigma^* = \{\epsilon, 0,1,00,01,\ldots,111,101,\ldots\}$
  - $\Sigma^+ = \Sigma^* - \{\epsilon\} = \{0,1,00,01,\ldots\}$

Operations on Languages
Concatenation

- L, M are languages.

- Concatenation operation • (or nothing) between languages
  - $L \bullet M = LM = \{xy | x \in L \text{ and } y \in M\}$
  - $L\{\epsilon\} = \{\epsilon\}L = L$
  - $L\varnothing = \varnothing L = \varnothing$

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

Exponents and Union of Languages

- **Exponents of languages**
  - $L^0 = \{\epsilon\}$
  - $L^1 = L$
  - $L^2 = L \bullet L$
  - $L^n = L \bullet L^{n-1}$, $n >= 1$
- **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 of Languages

- **Closure**
  - $L^* = L^0 \cup L^1 \cup \ldots \cup L^\infty$
- **Positive closure**
  - $L^+ = L^1 \cup L^2 \cup \ldots \cup L^\infty$ $LL^* = L^* - \{\epsilon\}$, if ϵ not in L
  - $L^* = \{\epsilon\} \cup L^+$
- Example: $A = \{a,b\}$
  - $A^* = \{\epsilon,a,b,aa,ab,ba,bb,\ldots\}$
    = All possible sequences of a and b.
- A language over A is always a subset of A^*.

Small Language Exercise

See 00-LectureExercises

Regular expressions

- **Regular expressions** are used to describe simple languages, e.g. basic symbols, tokens.
  - Example: identifier = letter • (letter | digit)*

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

Rules for constructing regular expressions

- S is an alphabet,
  - the regular expression r describes the language L_r,
  - the regular expression s corresponds to the language L_s, etc.
- Each symbol in the alphabet S is a regular expression which denotes {a}.
  - * = repetition, zero or more times.
  - + = repetition, one or more times.
  - . concatenation can be left out

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

Priorities

Highest	* +
	.
Lowest	|

Regular Expression Language Examples

- Examples: S = {a,b}
  - 1. r=a — L_r={a}
  - 2. r=a* — L_r={ϵ,a,aa,aaa, ...} = {a}*
  - 3. r=a|b — L_r={a,b}={a} ∪ {b}
  - 4. r=(a|b)* — L_r={a,b}*={ϵ,a,b,aa,ab,ba,bb,aaa,aab,...}
  - 5. r=(a*b*)* — L_r={a,b}*={ϵ,a,b,aa,ab,ba,bb,aaa,aab,...}
  - 6. r=a|ba* — L_r={a,b,ba,baa,baaa,...}={a or ba^i | $i \geq 0$}

- NB! {$a^n b^n$ | n>=0} cannot be described with regular expressions.
  - r=a*b* gives us Lr={a^i b^j | i,j>=0} does not work.
  - r=(ab)* gives us Lr={$(ab)^i$ | i>=0}={ϵ,ab,abab, ... } does not work.
- Regular expressions cannot ”count” (have no memory).

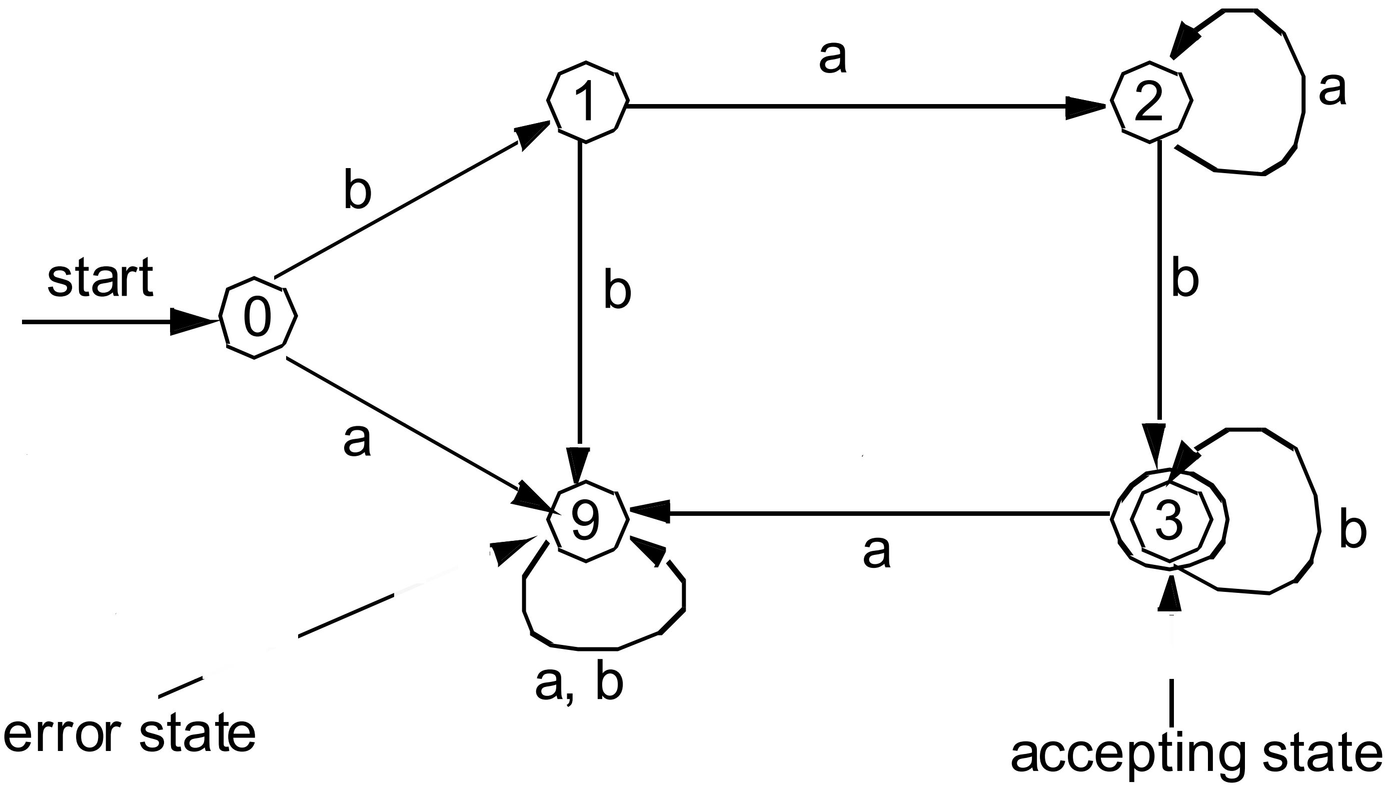
Finite state Automata and Diagrams

- (*Finite automaton*)
- Assume:
  - regular expression RU **= ba^+b^+ = baa ... abb ... b**
  - **L(RU) = { ba^nb^m | n, m $\geq$ 1 }**

- **Recognizer**
  - A program which takes a string x and answers yes/no depending on whether x is included in the language.
  - The first step in constructing a recognizer for the language L(RU) is to draw a state diagram (transition diagram).

State Transition Diagram

- state diagram (DFA) for ba^nb^m



Interpret a State Transition Diagram

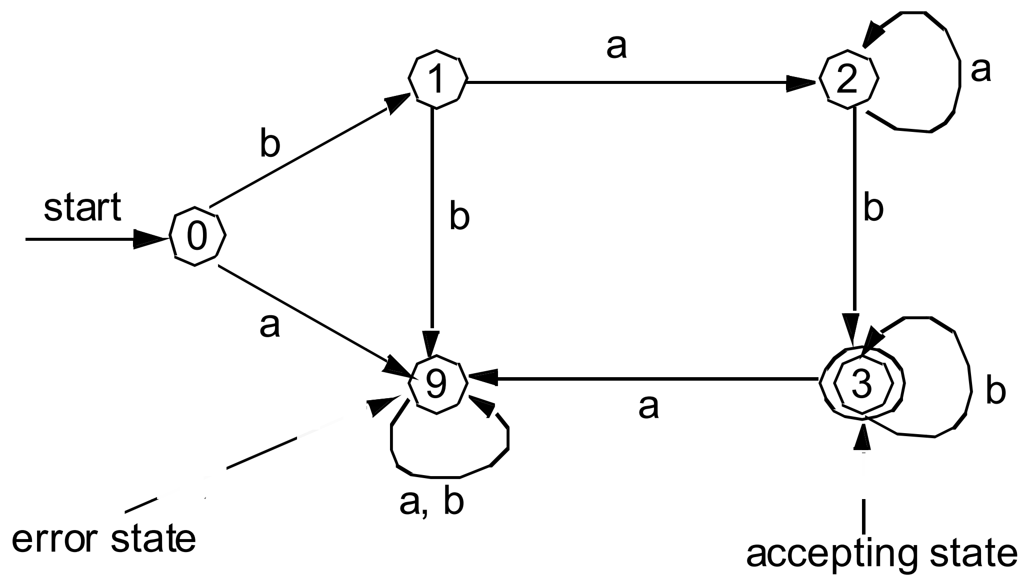
- Start in the starting node 0.
- Repeat until there is no more input:
  - Read input.
  - Follow a suitable edge.
- When there is no more input:
  - Check whether we are in a final state. In this case accept the string.
- There is an error in the input if there is no suitable edge to follow.
  - Add one or several error nodes.

Input and State Transitions

- Example of input: **baab**

- Then *accept* when there is no more input and state 3 is an accepting state.

Step	Current State	Input
1	0	baab
2	1	aab
3	2	ab
4	2	b
5	3	ϵ



Representation of State Diagrams by Transition Tables

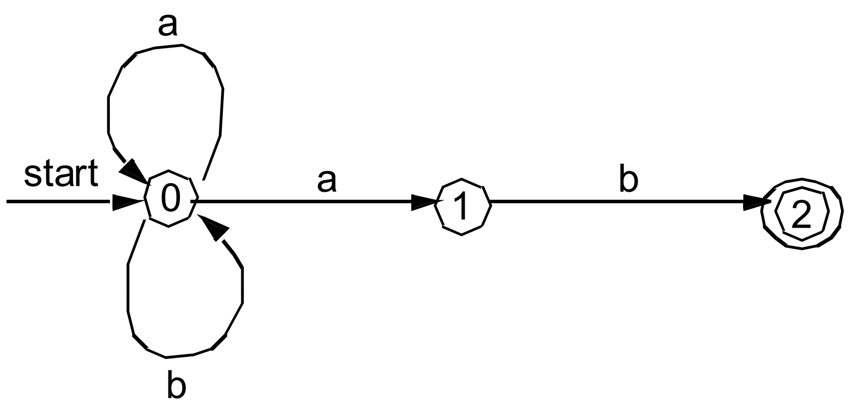
- The previous graph is a DFA (*Deterministic Finite Automaton*).
- It is deterministic because at each step there is exactly one state to go to and there is no transition marked "ϵ".
- A regular expression denotes a regular set and corresponds to an NFA *(Nondeterministic Finite Automaton).*

State	Accept	Found	Next state a	Next state b
0	no	ϵ	9	1
1	no	b	2	9
2	no	ba^+	2	3
3	yes	ba^+b^+	9	3
9	no			9

Transition Table
(Suitable for computer representation).

NFA and Transition Tables

Example: NFA for **(b|a)* ab**



state diagram for **(b|a)*ab**

state	a	b	Accept
0	{0,1}	{0}	no
1		{2}	no
2			yes

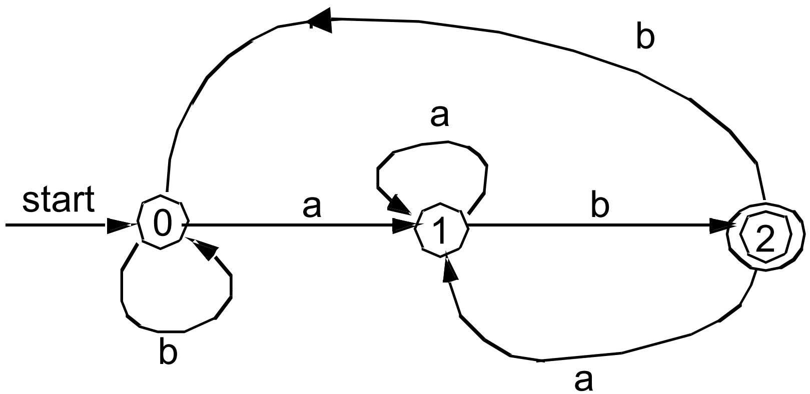
Transition table for **(b|a)*ab**

It requires more calculations to simulate an NFA with a computer program, e.g. for input **ab**, compared to a DFA.

Transforming NFA to DFA

- **Theorem**
  - Any NFA can be transformed to a corresponding DFA.
- When generating a recognizer automatically, the following is done:
  - regular expression $\rightarrow$ NFA.
  - NFA $\rightarrow$ DFA.
  - DFA $\rightarrow$ minimal DFA.
  - DFA $\rightarrow$ corresponding program code or table.

DFA for **(b|a)*ab**



Small Regular Expression and Transition Diagram/Table Exercise

See 00-LectureExercises