Lesson 1

Introduction to formal languages and automata theory

Agenda

- Hour I
  - Brief introduction to Automata Theory and Formal languages
  - Some hints for Lab 1
- Hour 2
  - Problem solving (See exercises on the course homepage)

A Formal Language

- Consists of words
  - A.k.a Strings, Symbol sequence?
- A word consists of letters
  - A.k.a Symbols, Glyphs
  - Do not need to be what we think about as letters
- Must be well-formed
- Classes of languages exist
  - More about this later in the course

What is a Letter and an Alphabet?

- Letter (Symbol, Glyph,…)
- Alphabet usually denoted with the Greek letter big sigma
- EX:
  - $\Sigma = \{A, B\}$

Word and Words

- From these definitions we know that AAA, ABA and ABBBBA are words in our language, assuming it is well formed
- A formal language is the set of the possibility infinite words we can construct from our alphabet

What is Automata?

Examples of Automata

- Your Computer
  - It is a Turing machine
- The Coffee Machine
  - Finite State Machine (FSM). However, might as well be a Turing machine as well ☹
- Different classes of Automata
  - Read more in introduction to Automata Theory, Languages, and Computation ☺
  - Chapter 1 & Chapter 2 are relevant for this course. Focus on concepts not proofs/lemmas.
  - ***Formal Languages and Automata Theory, 6 credits (TDDD14)***
- For Lab 1 we deal with FA and regular languages. More specifically regular expressions which we use to specify our Automata that does tokenizing!

What is Automata Theory?

- It is the theory/study of Automata
- In textbooks Automata usually looks like transition diagrams
- Moore and Mealy machines are a variant of automata with output ☺
  - Moore
    - The Output is associated with state
  - Mealy
    - The Output is associated with transition from one state to the next

Digital Logic

- Alphabet
  - $\Sigma=\{0,1\}$
- Words:
  - $\{01,10,11\ldots\}$

State Diagrams/Finite Automaton (FA)

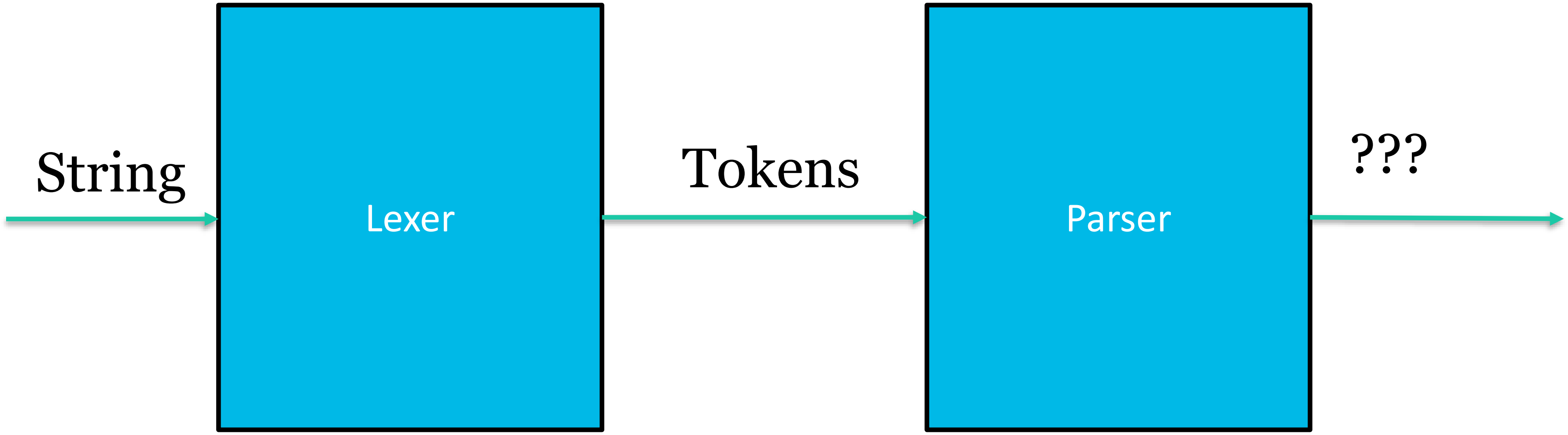
- Directed graph (Digraph)
  - Set of states:
  - Set of transitions:
- A string is accepted by a **FA** if we go from the start state to some accepted state
- Nondeterministic finite automaton(NFA)

Practical Applications

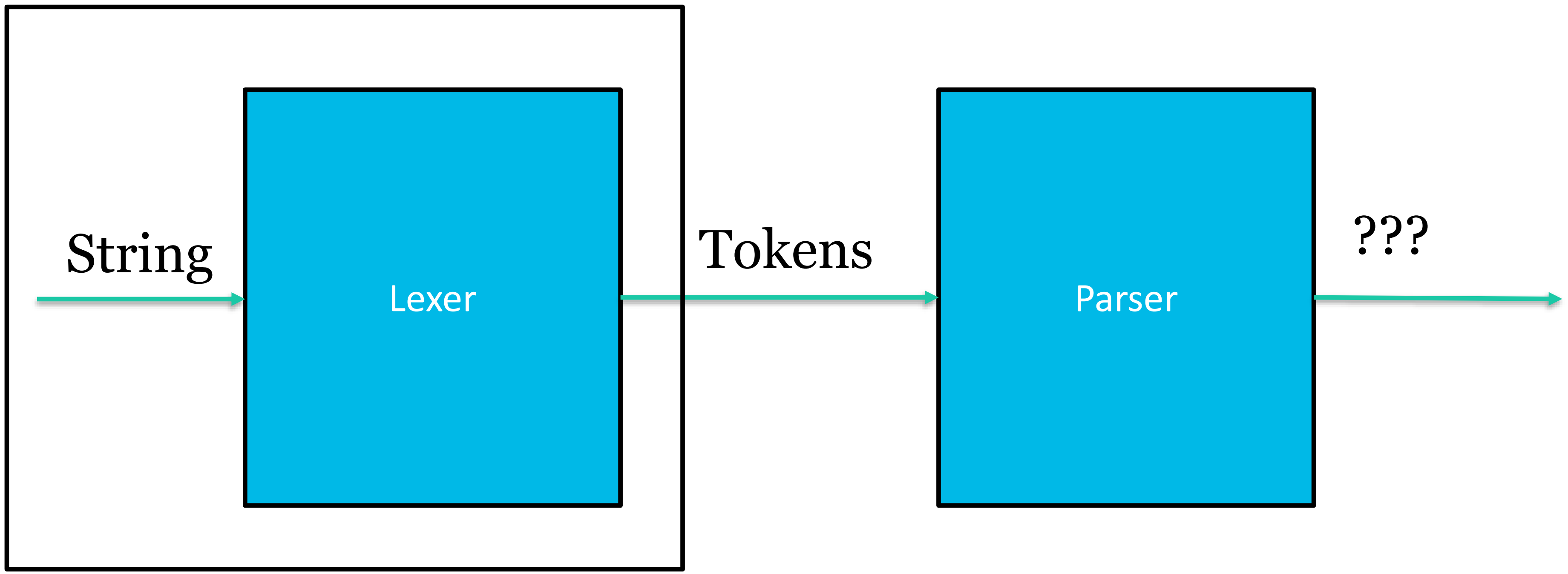
Applications

- Regular expressions
- Digital circuits
- Computers
- Compilers

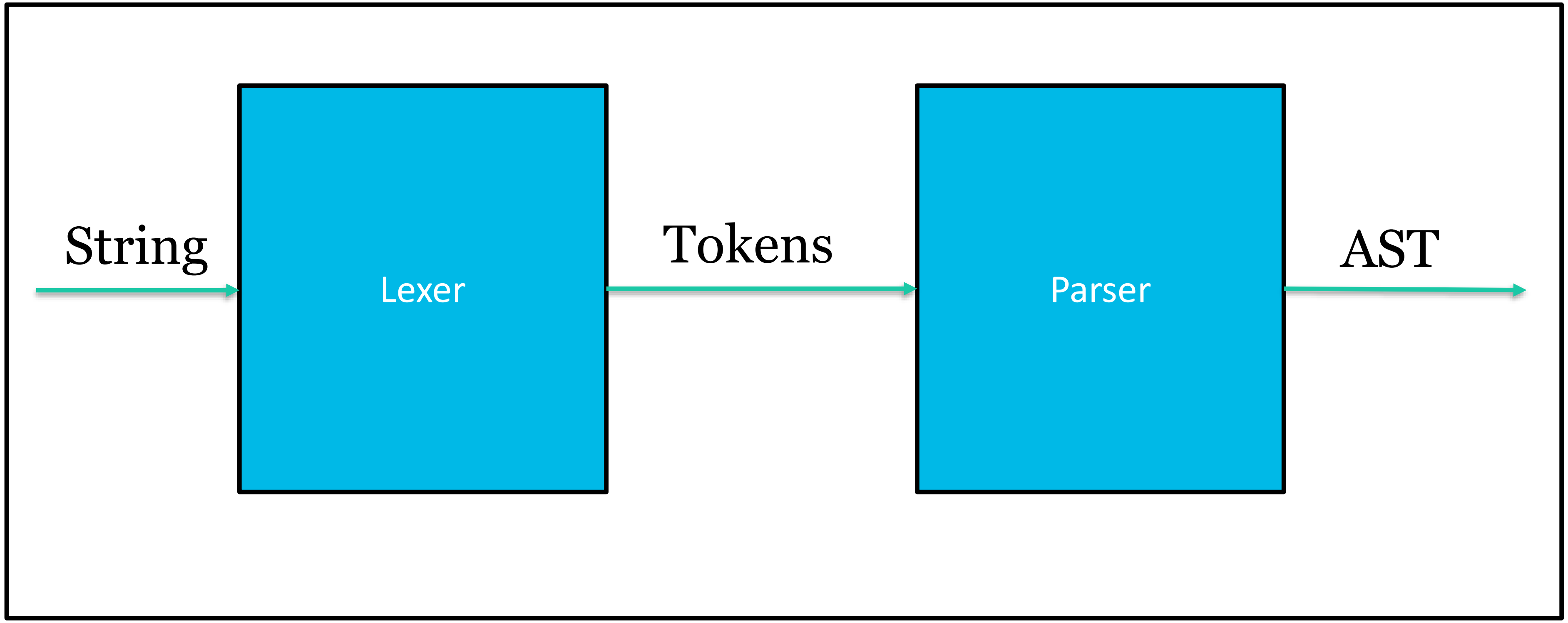
Compiler Pipeline



Lab 1



Lab 2



Regular expressions and FA

- What is accepted by **FA** can also be described by a regular expression!
- Important. The limitations of finite automata also applies to regular expressions
- Finite automata can only count.
- You can’t parse using regular expressions!

Dictionary. Some short terms explained

- $\boldsymbol{\Sigma}$ = Alphabet, sequence of symbols (Big Sigma)
- $\mathbf{Q}$ = The set of states in our FA
- $\boldsymbol{\delta}$ = State transition function (Little delta)
- F = Set of final states, or you can say accept states
- q_0 = Initial state
- FA = Finite Automata
- NFA = None deterministic finite automata
- DFA = Deterministic finite automata

Dictionary. Some short terms explained

- ε = Empty string (Small Epsilon)
  - AεBεC ⇔ ABC
- * = The Kleene star
- AB = Juxtaposition or concatenation between string A and B
- + = |
  - In the tradition of the text (Formal languages): + means or (|)
  - It might also mean concatenation/juxtaposition in some literature
    - Please state what definition you use!

Hints for Lab 1

Hints for Lab 1

- Instructions:
  - https://www.ida.liu.se/~TDDD55/laboratories/instructions/lab1.html
- Clone the lab from
  - https://gitlab.liu.se/tddd55/tddd55-lab
- It is important to consult the documentation and not attempt to make progress by trial and error!

Hints for Lab 1

- Lab 1 consists of several files
  - main.cc
  - Makefile
  - Makefile.dependencies
  - scanner.h
  - scanner.l
- scanner.l is the only file that you need to modify

Hints for Lab 1

- To Compile:
  - Type make at the directory where the files are
- Test the lab by executing:
  - ./scanner ./test/<file you want to run>

Hints for Lab 1

- Scanner specification via regular expressions
- Some definitions that usually means the same thing
  - Tokenizer, Lexical analyser, Scanner
- Necessary to escape special tokens (Or rather token that has a meaning in Flex)
- Try the examples from the Flex manual
  - https://www.ida.liu.se/~TDDB44/laboratories/instructions/_static/flex/index.html

Hints for Lab 1

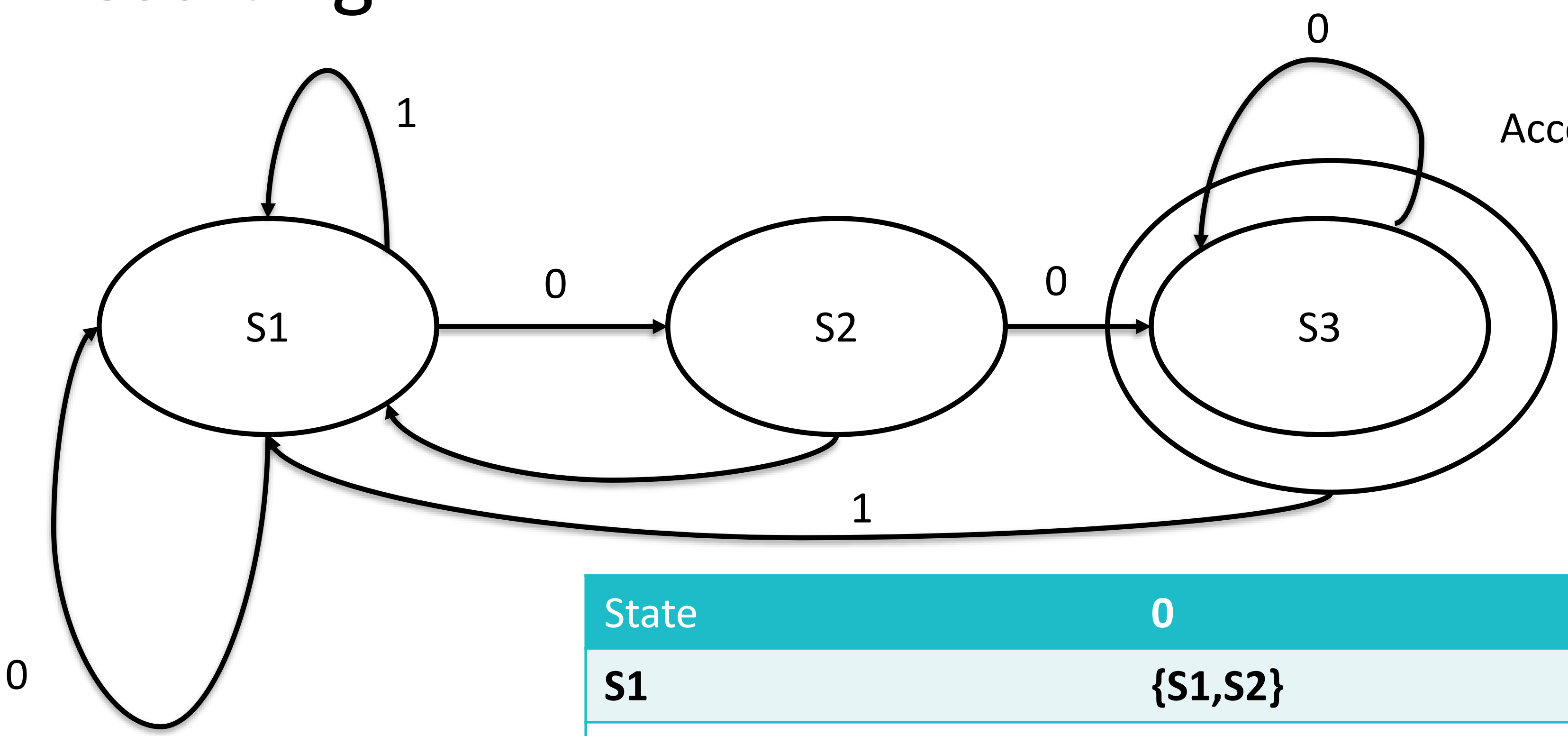
- An Integer with a dot
  - INTDOT [0-9]+\.
- An Integer
  - INTEGER [0-9]+
- A Integer or an an Integer with a dot
  - INTEGER_OR_INTDOT (INTEGER)|(INTDOT)
- Nested comments might be hard. **Tip:** Read up on Flex start conditions. See chapter 10 in the flex manual.

Extended solution proposals

Extended solution proposal to Exercise 2.4

- We can write L1 as the following regular expression:
  - (0|1)*00
- From this we define our NFA
  - From our start state we can select between two paths
    - 0* or 1*
  - For 00. We simply go forward two steps

Resulting NFA



Note not according to Thompsons algorithm

Here, I have removed ε moves for clarity!

State	0	1
S1	**{S1,S2}**	**S1**
S2	**S3**	**S1**
S3	**S1**	**S2**

Extended solution proposal to Exercise 2.4

- Deriving a DFA
- $(Q , \Sigma , \delta , q_0 , F)$
  - Alphabet: Σ = {0, 1}
  - Transition function: **δ** See next slide
  - States: Q = {S1,S2,S3} //*Intuition*: We have to handle atleast 3 tokens
  - Accept states: F = {S3}
    - $S1 -> S2 -> S3$ for the input 00

State transition table for our transition function: δ

State	0	1
S1	**S2**	**S1**
S2	**S3**	**S1**
S3	**S3**	**S1**

Alphabet: Σ = **{0, 1}**
Transition function: $\boldsymbol{\delta}$ The state transistion table
States: **Q = {S1,S2,S3}**
Accept states: **F = {S3}**

Extended solution proposal for 6.1

- Given the following regular expressions
  - $00(1|0)^* 1$
  - $101(101)^* 101(010)^*$
  - $(11|010)^* 11(00|11)^*$
- Find Context Free Grammars(CFG) that correspond to the word that is accepted by the regular expression
- If there are any insecurities regarding CFG and production rules
  - See lecture 3

Regular Expression 6.1 A

- $00(1 \mid 0)^* 1$
- For this expression we shall first consider the types of strings we can accept
- We know that our alphabet is:
  - Σ={0,1}
- Let’s derive a set of words that we would accept:
  - {001, 0001, 00101}
- Note the Kleene star * and |
  - Kleene star * allows the empty string

Deriving a CFG for 6.1 A

- $00(1|0)^* 1$
- Intuition
  - From the expression above we notice that we always need 00 as a prefix
  - Likewise the suffix must be 1
- Rule 1
  - S → $00A1$
  - We do not yet bother with what A should be

Deriving CFG for 6.1 A

- $00(\mathbf{1}|\mathbf{0})^{*}\ 1$
  - We know that $1|0$ means a 1 or zero
  - From this we know that $(\mathbf{1}|\mathbf{0})$* gives the set:
    - {ε ,0,1,00,01,10,11,001,...}
    - 2^{N} Different combinations where N is positive infinity
- Zero (ε) times gives us:
  - 001 ⇔ 00ε1 So we can introduce the rule A → ε

Deriving Rules for 6.1 A

- $S \rightarrow\ 00A1$ & $A\ \rightarrow\ \varepsilon$
  - Now we look at $00(\mathbf{1}|\mathbf{0})^*\ 1$ again
  - $(\mathbf{1}|\mathbf{0})^*$ //{ε, 10, 110,11110,...}
- For the entire expression we would have 00101 for 10
  - Notice that we need flexibility here. We can't simply state that A is 10. The reason is that A might be **110** or **1110**
  - If we say $A \rightarrow 1A$ ***or*** $A \rightarrow 0A$ we get this flexibility
- Set of production rules for our CFG are: $\mathbf{\{S \rightarrow\ 00A1,\ A\ \rightarrow\ \varepsilon,\ A \rightarrow 1A\ ,A \rightarrow 0A\}}$

John Tinnerholm
Jonas Wallgren

www.liu.se

LINKÖPING
UNIVERSITY

References

Hopcroft, J. E. (2008). *Introduction to automata theory, languages, and computation*. Pearson Education India.

Aho, A. V., Sethi, R., & Ullman, J. D. (1986). Compilers, principles, techniques. *Addison wesley*, *7*(8), 9.