TDDD55-Lesson 1
Introduction to formal languages and automata theory and a brief introduction to Lab 1

John Tinnerholm
Jonas Wallgren

LINKÖPING UNIVERSITY

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)
  - If time permits, start with Lab-1

Image generated by OpenAI's DALL-E through ChatGPT

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
  - That is, in short, conform to the rules and structures defined for it
  - Natural Language Examples:
    - ❑ I painted a nonexistent house
      - ❖ Well-formed but does not make sense
    - ❑ bicycle ride I
      - ❖ Not well-formed but can be understood
- Note, theoretical terms but you may encounter them when you look for literature yourself
- Classes of languages exist
  - More about this later in the course

Image generated by OpenAI's DALL-E through ChatGPT

What is a Letter and an Alphabet?

- Letter (Symbol, Glyph,…)
- Alphabet usually denoted with the Greek letter sigma
- EX:
  - $\Sigma = \{A, B\}$
  - $\Sigma = \{1,2,3,4\}$
  - $\Sigma = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, \ldots\ ,\ Å, Ä\ Ö\}$

Image generated by OpenAI's DALL-E through ChatGPT

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
- EX:
  - $\Sigma = \{A, B\}$
- Some possible words
  - W = {A, AA, AAA, AAAA, ….}

Automata?

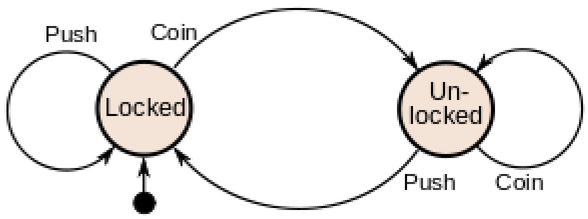
Examples of Automata

- Your Computer
  - It is a Turing machine
- The Coffee Machine
  - Finite State Machine (**FSM**). However, it might as well be a Turing machine as[1] ☹
- 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[2].
  - Full Course
    - ❑ 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 do tokenizing

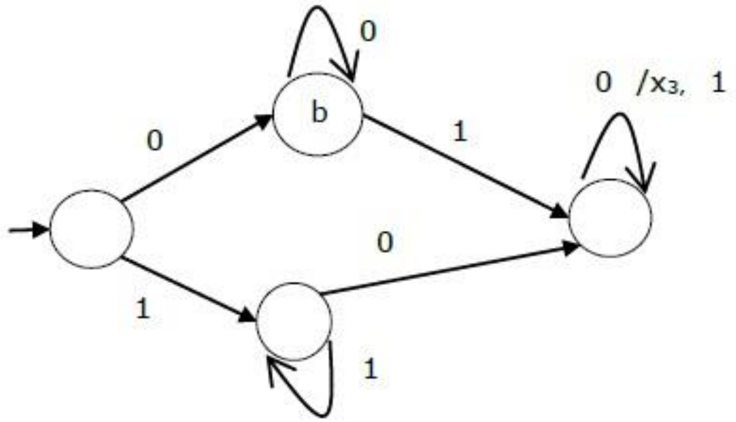
[1]Now, in the 2020s, even fridges might be Turing Machines

[2]This course focuses on the practical application of these concepts

- What is Automata Theory?



This Photo by Unknown Author is licensed under CC BY-SA



This Photo by Unknown Author is licensed under CC BY-SA

- It is the theory/study of Automata
- In textbooks, Automata are usually depicted using State Diagrams
- Moore and Mealy machines are a variant of automata with output ⍰
  - Moore
    - ❑The Output is associated with the state
  - Mealy
    - ❑The Output is associated with the transition from one state to the next

Digital Logic

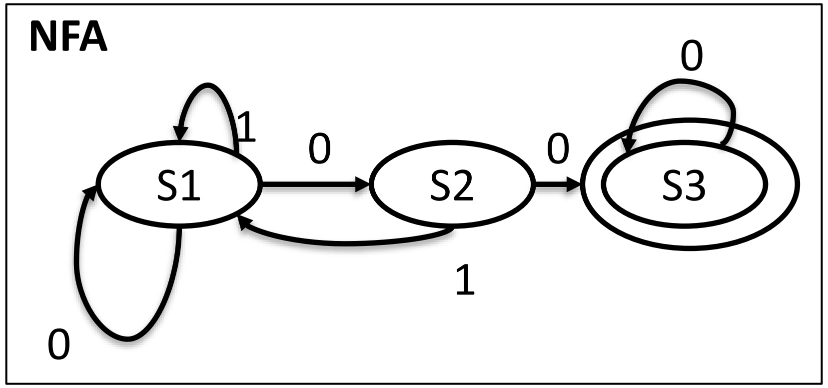
- Digital Logic is a language
- Alphabet
  - $\Sigma=\{0,1\}$
- Words:
  - {00, 01,10,11…}
- Can be described by a regular expression
  - $[0\text{-}1]^+$

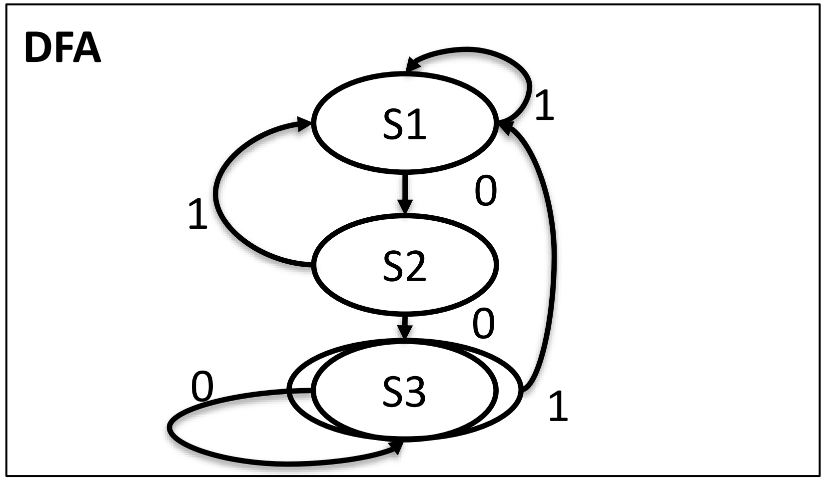
State Diagrams/Finite Automaton (FA)

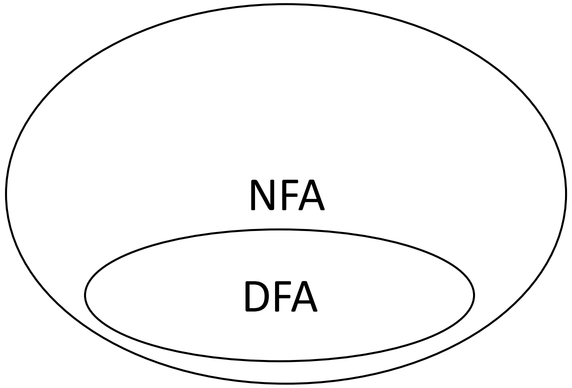
- Directed graph (Digraph)
  - Set of states:
  - Set of transitions:
- A string is accepted by an **FA (Finite Automaton)** if we go from the start state to some accepted state
- Nondeterministic finite automaton **(NFA)**
  - Theoretical
  - Can be simulated

State Diagrams/Finite Automaton (FA)

- Nondeterministic finite automaton **(NFA)**
  - Theoretical
  - Can be simulated
- Deterministic Finite Automaton **(DFA)**
- Furthermore
  - A DFA is an NFA, but a DFA is not an NFA
  - DFA can be seen as a “restricted” NFA
  - NFA gives you more creative freedom when modeling
    - Ad-hoc Transitions







Practical Applications

Applications

REGULAR EXPRESSIONS

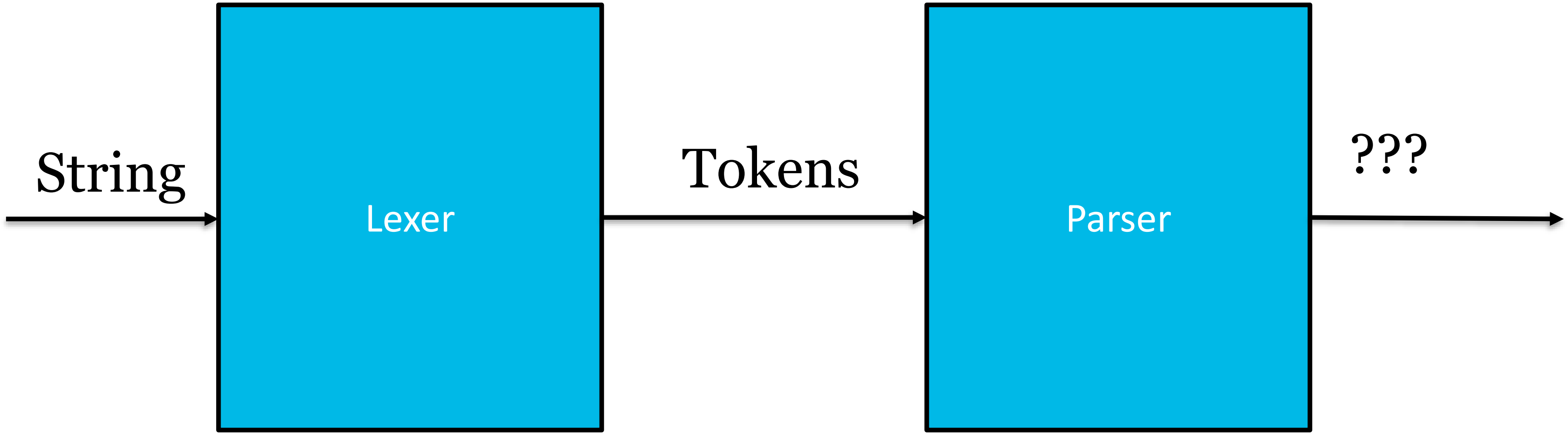
DIGITAL CIRCUITS

COMPUTERS

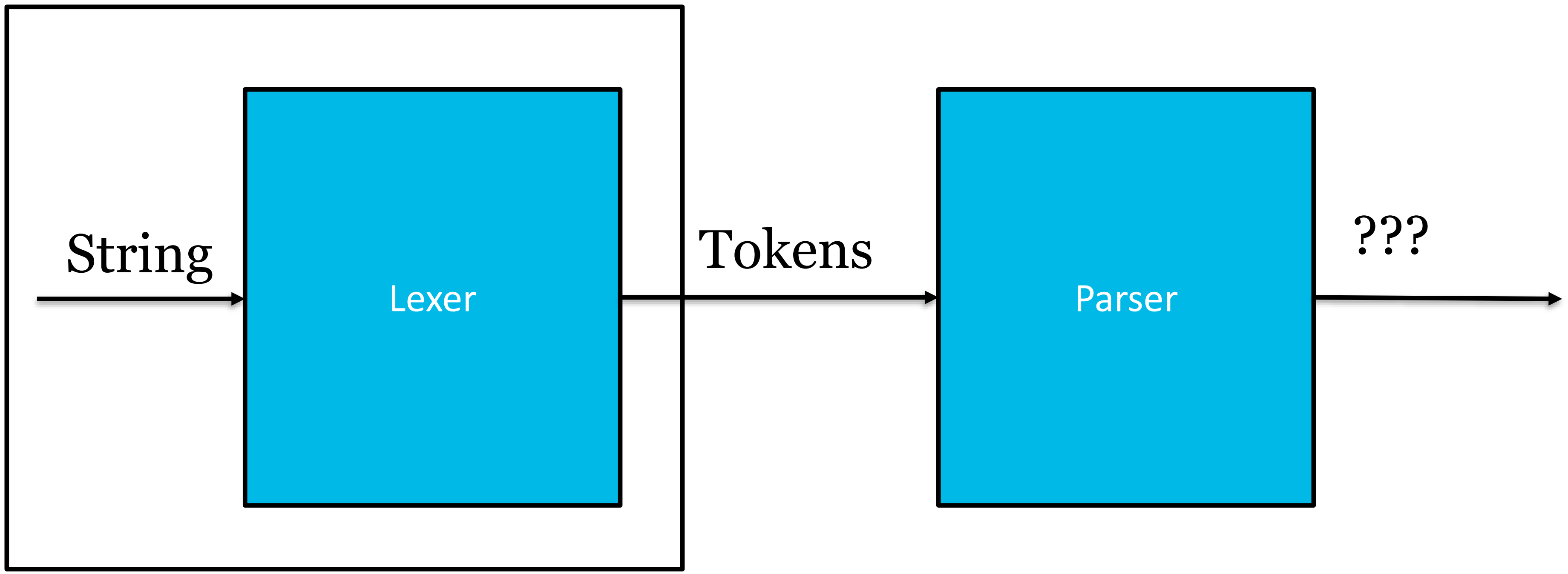
COMPILERS

…

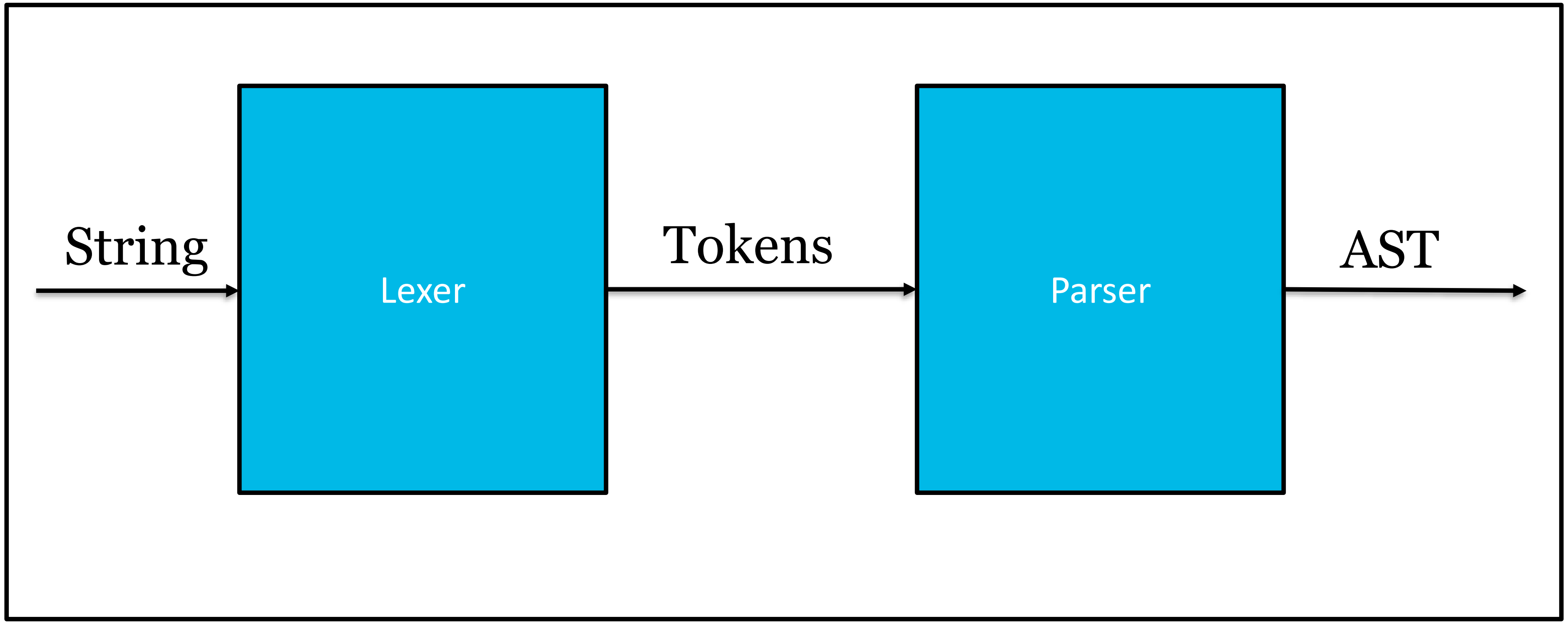
Compiler Pipeline



Lab 1



Lab 2



Regular expressions and Finite Automata (**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
    - Can not solve the problem of balanced parenthesis
    - Can not process context free-grammars
  - Hence, you can not parse using regular expressions

Dictionary. Some short terms explained

- **Σ** = Alphabet, sequence of symbols (Sigma)
- **Q** = The set of states in our FA
- ***δ*** = State transition function (Small 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 (Concatenation) between string A and B
- + and |
  - In the tradition of the text (Formal languages): + means "or" (|)
  - It might also mean concatenation/juxtaposition in some recent literature
    - Please state what definition you use

Thompsons Algorithm: A technique to generate NFA from a Regular Expression

Converting Regular Expresisons to NFA: **Thompsons algorithm**

- Converts regular expressions into a corresponding NFA
- **Not a mandatory part of the course**. However, it might be useful to learn this algorithm anyway
  - Usually, *intuitive* approaches work as well

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
- In this course, it is extra important to consult the documentation and not attempt to make progress by trial and error!
- Remember also to handle tabs (\t)
  - **New in 2023**
    - ❑ A test for Lab-1 that checks this

> ✓ ***Additional information is available in the README for each Lab***
> ✓ See the Skeleton — TDDD55 Compilers and Interpreters documentation for a complete description of the entire lab project

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***
  - You will reuse your results later in Lab 3/4

Hints for Lab 1

- To Compile:
  - Type make in the directory where the files are
- Test the lab by executing:
  - ./scanner ./test/<file-you-want-to-run>
- It is recommended that you start with the identifiers

Hints for Lab 1

- Scanner specification via regular expressions
- Some definitions that usually mean the same thing
  - Tokenizer, Lexical analyzer, 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
- Remember also to handle tabs (!) ”**t**”

Hints for Lab 1

- An Integer with a dot
  - INTDOT $[0\text{-}9]^{+}\backslash.$
- An Integer
  - INTEGER $[0\text{-}9]^{+}$
- An Integer without a period or an Integer with a period
  - INTEGER_OR_INTDOT (INTEGER)|(INTDOT)
- Nested comments might be hard.
  - **Tip:** Read up on Flex start conditions.
  - See chapter 10 in the flex manual.

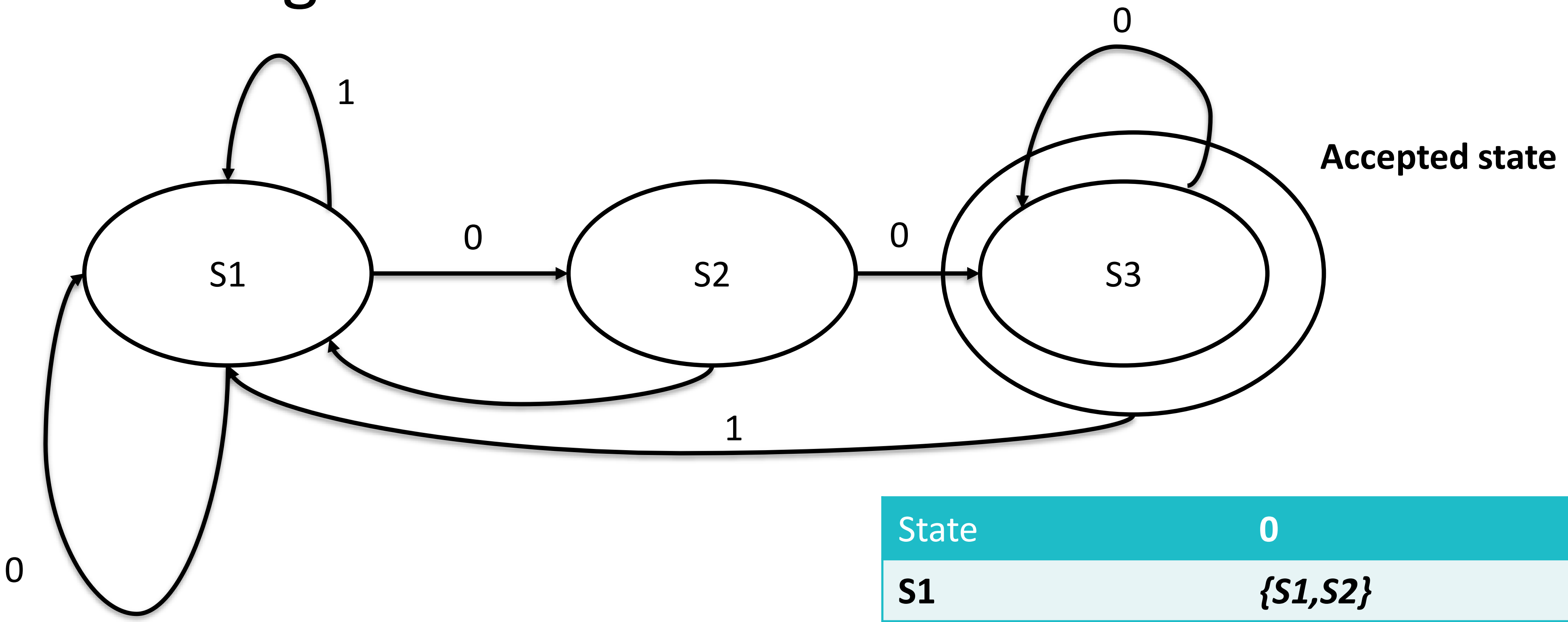
Extended solution proposals for 2.1 (A) and 2.4

Extended solution proposal to Exercise 2.1 (A)

a) $L_1 = \{x \in \{0,1\}^* \mid x \text{ ends in } 00\}$

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

Resulting NFA



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

Original regular expression
(0|1)*00

Note not according to Thompson's algorithm. Furthermore, note that this is not a DFA, it is non-deterministic since we can either go to S1 or to S2 in state S1

Extended solution proposal to Exercise 2.1 (A)

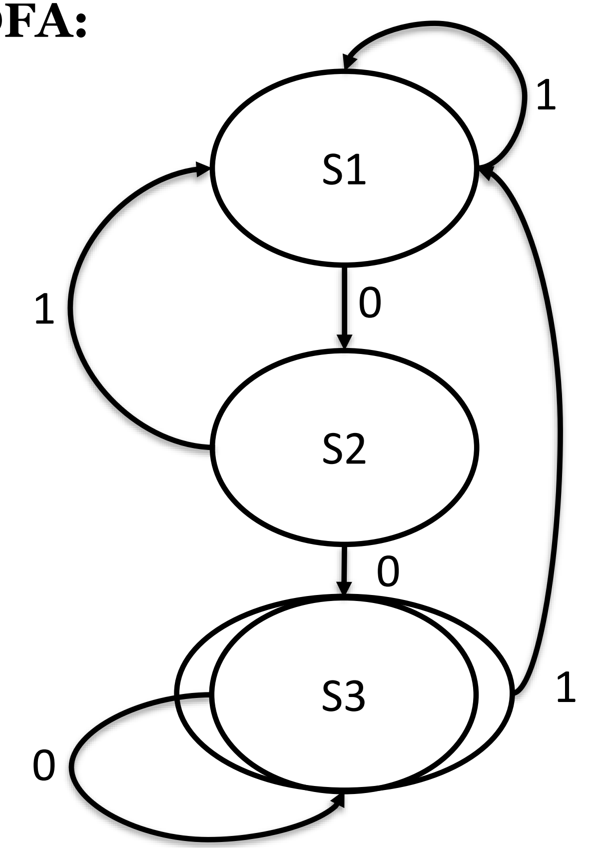
- Deriving a DFA
- $(Q , \Sigma , \delta , q_0 , F)$
  - Alphabet: Σ = {0, 1}
  - Transition function: $\boldsymbol{\delta}$ See next slide
  - States: Q = {S1,S2,S3} //*Intuition*: We have to handle atleast 3 tokens
  - Accept states: F = {S3}
    - $S1 -> S2 -> S3$ for the input 00

State transition table for our transition function: δ

State \ Input	0	1
S1	S2	S1
S2	S3	S1
S3	S3	S1

- Alphabet: Σ = **{0, 1}**
- Transition function: $\boldsymbol{\delta}$
- States: **Q = {S1,S2,S3}**
- Accept States: **F = {S3}**

Resulting DFA:



Teaser for the next theme: (Context Free Grammars/ Lab 2)

- Grammars
- Describing Regular Expressions using Grammar

Regular Expressions to Grammar (Exercise 2.4)

- 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 2.4 (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 2.4 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 a CFG for 2.4 A

- $00(\mathbf{1}|\mathbf{0})^{*}\ 1$
  - We know that 1|0 means a 1 or zero
  - From this we know that (**1**|**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 2.4 (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: $\{\boldsymbol{S \rightarrow 00A1,\ A \rightarrow \varepsilon,\ A \rightarrow 1A\ ,\ A \rightarrow 0A}\}$

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.