IN2080

1. Introduction & Regular Languages

Einar Broch Johnsen

University of Oslo

31st January 2023



Department of Informatics



University of Oslo

Details on the Course

- Two parts:
  - Computability theory (first half of semester, held by Einar Broch Johnsen), and
  - Complexity theory (second half, held by Lars Kristiansen)

Details on the Course

- Two parts:
  - Computability theory (first half of semester, held by Einar Broch Johnsen), and
  - Complexity theory (second half, held by Lars Kristiansen)
- The course will closely follow Michael Sipser's book

  Introduction to the Theory of Computation (3rd International Edition)

  both in course content and exercises

Details on the Course

- Two parts:
  - Computability theory (first half of semester, held by Einar Broch Johnsen), and
  - Complexity theory (second half, held by Lars Kristiansen)
- The course will closely follow Michael Sipser's book

  Introduction to the Theory of Computation (3rd International Edition)

  both in course content and exercises
- Prerequisites are IN1150 and chapter 0 of the book
  (very brief and incomplete refresher soon)

Details on the Course

- Two parts:
  - Computability theory (first half of semester, held by Einar Broch Johnsen), and
  - Complexity theory (second half, held by Lars Kristiansen)
- The course will closely follow Michael Sipser's book

  Introduction to the Theory of Computation (3rd International Edition)

  both in course content and exercises
- Prerequisites are IN1150 and chapter 0 of the book
  (very brief and incomplete refresher soon)
- Remember: lectures are useful, but doing exercises yourself is the most important!
  → **group exercises** (held by Mick Guiotto Williams)

Course Structure for Part 1 Computability Theory

For the first half of the course:

- **Tuesday lecture:** new theory and material
- **Wednesday lecture:** can also feature new theory and material, but mostly reserved for in-depth discussions, examples, open Q&A

Course Structure for Part 1 Computability Theory

For the first half of the course:

- **Tuesday lecture:** new theory and material
- **Wednesday lecture:** can also feature new theory and material, but mostly reserved for in-depth discussions, examples, open Q&A

- Do you have questions outside of the lectures?
  $\rightarrow$ group exercises, Mattermost(?)

Course Structure for Part 1 Computability Theory

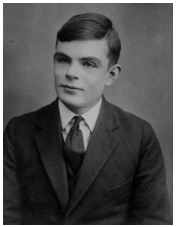
For the first half of the course:

- **Tuesday lecture:** new theory and material
- **Wednesday lecture:** can also feature new theory and material, but mostly reserved for in-depth discussions, examples, open Q&A

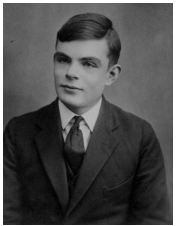
- Do you have questions outside of the lectures?
  $\rightarrow$ group exercises, Mattermost(?)

- **Book:** Sipser
- **YouTube** videoer fra Sipser [her]
- **Slides**: link under "Timeplan" on the course page

So what's it all about?

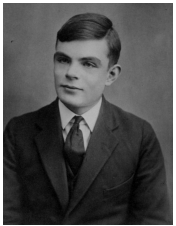


So what's it all about?



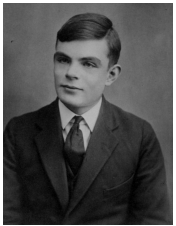
- Alan Turing
  (1912-1954)

So what's it all about?



- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

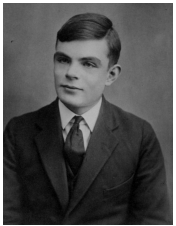
So what's it all about?



- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

$\rightarrow$ Turing machines

So what's it all about?

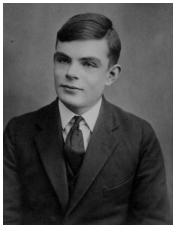


- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

$\rightarrow$ Turing machines



So what's it all about?



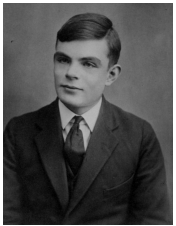
- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

→ Turing machines



- Noam Chomsky (1928-)
- "Father" of modern linguistics

So what's it all about?



- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

→ Turing machines



- Noam Chomsky (1928-)
- "Father" of modern linguistics
- classification of formal languages

→ Chomsky hierarchy

So what's it all about?

- What is an "algorithm"?

So what's it all about?

- What is an "algorithm"?
- Automata and formal languages (part 1 of the course)
  - programs with *very* limited memory
  - programs to recognize/generate "words" in a language
  - typically used in compilers

So what's it all about?

- What is an "algorithm"?
- Automata and formal languages (part 1 of the course)
  - programs with *very* limited memory
  - programs to recognize/generate "words" in a language
  - typically used in compilers
- Turing machines (part 2 of the course)
  - programs with unrestricted memory
  - how expensive is a program?
  - $\rightarrow$ complexity

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**:

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**:
  - $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**:
  - $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and
  - $\overline{A \cap B} = \bar{A} \cup \bar{B}$

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**:
  - $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and
  - $\overline{A \cap B} = \bar{A} \cup \bar{B}$
- **Power Set**: $\mathcal{P}(A) = \{S \mid S \subseteq A\}$

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**:
  - $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and
  - $\overline{A \cap B} = \bar{A} \cup \bar{B}$
- **Power Set**: $\mathcal{P}(A) = \{S \mid S \subseteq A\}$
  - Example: $\mathcal{P}(\{0, 1\}) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

The Basics

- **Tuple**: ordered collection of objects
  - $(a, a, b) \neq (a, b)$
- **Cartesian product**: $A \times B = \{(a, b) \mid a \in A, b \in B\}$
- **Function**: $f : A \to B$.
  - Assigns to each element $a \in A$ a unique element $f(a) \in B$.

Strings

- Let Σ be an alphabet (non-empty finite set of symbols)
- A **string** is a finite sequence of symbols from Σ
- Empty string ε

Strings

- Let Σ be an alphabet (non-empty finite set of symbols)
- A **string** is a finite sequence of symbols from Σ
- Empty string ε

- **Example:** *abracadabra* is a string over the alphabet $\Sigma = \{a, b, c, \ldots, z\}$

Strings

- Let Σ be an alphabet (non-empty finite set of symbols)
- A **string** is a finite sequence of symbols from Σ
- Empty string ε

- **Example:** *abracadabra* is a string over the alphabet $\Sigma = \{a, b, c, \ldots, z\}$

- A **language** = set of strings over an alphabet

Finite Automata

- Computational model of a computer with finite memory

Finite Automata

- Computational model of a computer with finite memory
- Takes an input string w and decides whether to *accept* or *reject*

Finite Automata

- Computational model of a computer with finite memory
- Takes an input string w and decides whether to *accept* or *reject*
- Can be used to answer such questions as
  - "Is w a palindrome?"
  - "Is w a valid program in a given programming language?"

Finite Automata

- Computational model of a computer with finite memory
- Takes an input string w and decides whether to *accept* or *reject*
- Can be used to answer such questions as
  - "Is w a palindrome?"
  - "Is w a valid program in a given programming language?"
- Usually depicted as a graph for ease of reading:
  - Nodes represent *states* in which the automaton can be

Finite Automata

- Computational model of a computer with finite memory
- Takes an input string w and decides whether to *accept* or *reject*
- Can be used to answer such questions as
  - "Is w a palindrome?"
  - "Is w a valid program in a given programming language?"
- Usually depicted as a graph for ease of reading:
  - Nodes represent *states* in which the automaton can be
  - Edges between nodes represent the transition between states given a parsed input

Finite Automata

- Computational model of a computer with finite memory
- Takes an input string w and decides whether to *accept* or *reject*
- Can be used to answer such questions as
  - "Is w a palindrome?"
  - "Is w a valid program in a given programming language?"
- Usually depicted as a graph for ease of reading:
  - Nodes represent *states* in which the automaton can be
  - Edges between nodes represent the transition between states given a parsed input
  - Always exactly one start node: 

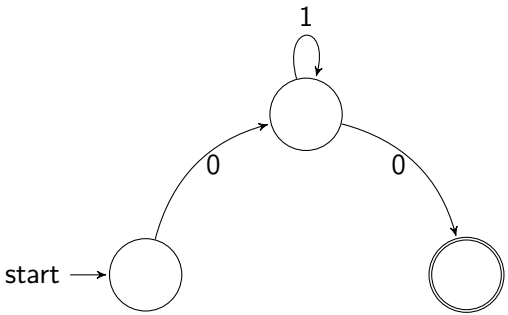
Finite Automata

- Computational model of a computer with finite memory
- Takes an input string w and decides whether to *accept* or *reject*
- Can be used to answer such questions as
  - "Is w a palindrome?"
  - "Is w a valid program in a given programming language?"
- Usually depicted as a graph for ease of reading:
  - Nodes represent *states* in which the automaton can be
  - Edges between nodes represent the transition between states given a parsed input
  - Always exactly one start node: 

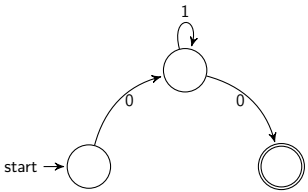
  - . . . as well as some accept states: 

Finite Automata

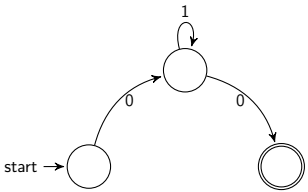
Example:



What does it mean for a finite automaton to "accept" an input w?

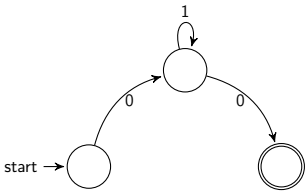


What does it mean for a finite automaton to "accept" an input w?



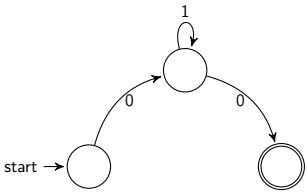
- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;

What does it mean for a finite automaton to "accept" an input w?



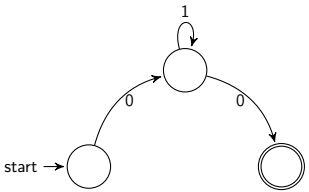
- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;

What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.

What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.
- the example automaton accepts all inputs, *words*, that start and end with 0, with only 1's in between.

Deterministic Finite Automata

Definition

A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of *states*

Deterministic Finite Automata

Definition

A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of *states*
2. Σ is a finite set called the *alphabet*

Deterministic Finite Automata

Definition

A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of *states*
2. Σ is a finite set called the *alphabet*
3. $\delta : Q \times \Sigma \to Q$ the transition function

Deterministic Finite Automata

Definition

A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of *states*
2. Σ is a finite set called the *alphabet*
3. $\delta : Q \times \Sigma \to Q$ the transition function
4. q_0 the start state

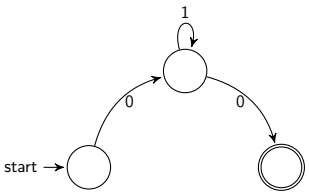
Deterministic Finite Automata

Definition

A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of *states*
2. Σ is a finite set called the *alphabet*
3. $\delta : Q \times \Sigma \to Q$ the transition function
4. q_0 the start state
5. $F \subseteq Q$ the set of accept states.

What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.
- the example automaton accepts all inputs, *words*, that start and end with 0, with only 1's in between.

Acceptance for Deterministic Finite Automata

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $\delta(s_i, w_{i+1}) = s_{i+1}$ (a valid transition is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

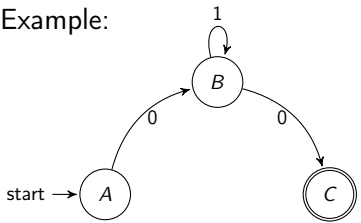
Acceptance for Deterministic Finite Automata

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $\delta(s_i, w_{i+1}) = s_{i+1}$ (a valid transition is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

Example:



For $w = w_1 w_2 w_3 w_4 w_5 = 01110$,

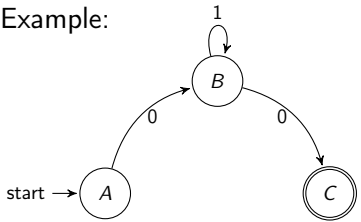
Acceptance for Deterministic Finite Automata

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $\delta(s_i, w_{i+1}) = s_{i+1}$ (a valid transition is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

Example:



For $w = w_1 w_2 w_3 w_4 w_5 = 01110$,
$s = s_0 s_1 s_2 s_3 s_4 s_5 = ABBBBC$

Regular Languages

Given an alphabet Σ a language L is a set of words $w = w_1 \cdots w_n$ such that each $w_i \in \Sigma$.

Regular Languages

Given an alphabet Σ a language L is a set of words $w = w_1 \cdots w_n$ such that each $w_i \in \Sigma$.

Definition

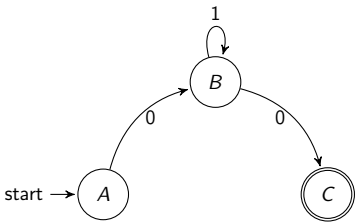
A language L is a *regular language* if there exists a DFA M that accepts each word in L, i.e., $L = \{w \mid M \text{ accepts } w\}$.

Regular Languages

Given an alphabet Σ a language L is a set of words $w = w_1 \cdots w_n$ such that each $w_i \in \Sigma$.

Definition

A language L is a *regular language* if there exists a DFA M that accepts each word in L, i.e., $L = \{w \mid M \text{ accepts } w\}$.



describes

$$L_1 = \{w \mid w \text{ starts and ends with 0, with only 1's in between}\}.$$

$\rightarrow L_1$ is regular

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: similarly, $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$.

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: similarly, $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$.
- Concatenation: $L_1 L_2 = \{w \mid w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\}$

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: similarly, $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$.
- Concatenation: $L_1 L_2 = \{w \mid w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\}$
- Kleene star: $L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0, \text{ each } x_i \in L_1\}$

Regular Languages

Theorem

The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

Theorem

The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

Proof idea: We multitask! Construct "product" automaton that runs both DFAs in parallel:

$(Q_1 \times Q_2, \Sigma, \delta, F)$ where

- $\delta((s_1, s_2), w_i) := (\delta_1(s_1, w_i), \delta_2(s_2, w_i))$

Theorem

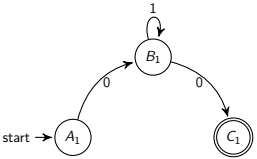
The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

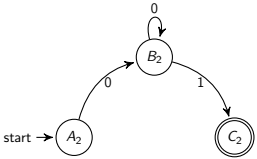
Proof idea: We multitask! Construct "product" automaton that runs both DFAs in parallel:

$(Q_1 \times Q_2, \Sigma, \delta, F)$ where

- $\delta((s_1, s_2), w_i) := (\delta_1(s_1, w_i), \delta_2(s_2, w_i))$
- $F = \{(s_1, s_2) \mid s_1 \text{ or } s_2 \text{ is an accepting state}\}$ for union,
- $F = \{(s_1, s_2) \mid s_1 \text{ and } s_2 \text{ is an accepting state}\}$ for intersection

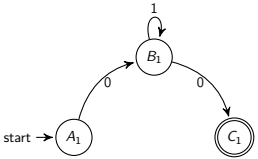
Union of regular languages: Example

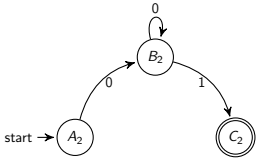




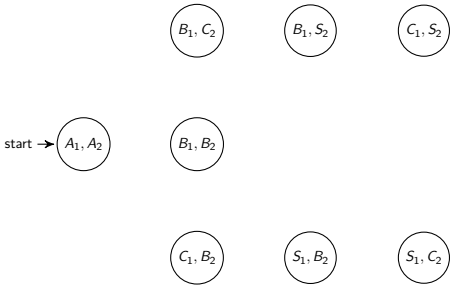
with implicit sink states S_1, S_2.

Union of regular languages: Example

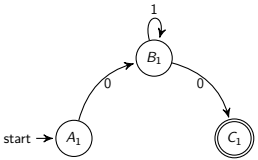


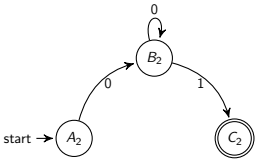


with implicit sink states S_1, S_2.

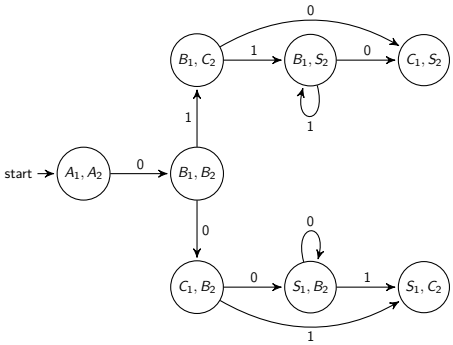


Union of regular languages: Example

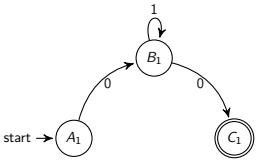


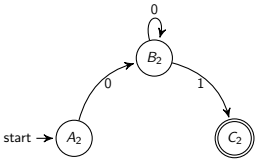


with implicit sink states S_1, S_2.

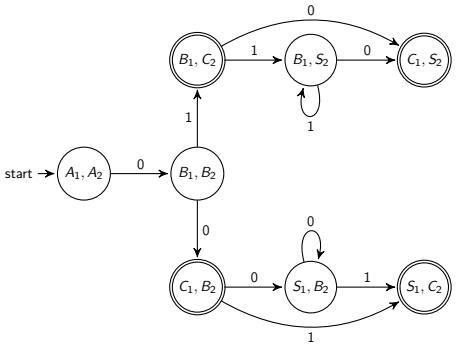


Union of regular languages: Example





with implicit sink states S_1, S_2.



Summary

- **Basics:** sets, de Morgan's laws, tuples, products, functions, strings
- **Finite automata:**
  - Accepts or rejects strings (so-called *words*)
  - States, transitions, start state, accept states
  - Formal definitions of DFA, acceptance, regular language
  - Operations on languages: union, intersection, concatenation, Kleene star
  - Theorem: RL closed under union and intersection

Summary

- **Basics:** sets, de Morgan's laws, tuples, products, functions, strings
- **Finite automata:**
  - Accepts or rejects strings (so-called *words*)
  - States, transitions, start state, accept states
  - Formal definitions of DFA, acceptance, regular language
  - Operations on languages: union, intersection, concatenation, Kleene star
  - Theorem: RL closed under union and intersection
- **Tomorrow:** Non-deterministic automata!