CPSC-406 Report

Ethan Tapia
Chapman University

May 10, 2025

Abstract

TBD

Contents

1 Introduction

2 Week by Week

2.1 Week 1

Lecture Summary

A finite automaton consists of a finite set of **states** (Q), an **alphabet** (Σ), a **transition function** (δ), a **starting state** (q_0), and a set of **accepting states** (F).

It can be formally represented as:

$$M = (Q, \Sigma, \delta, q_0, F)$$

where:

- Q is the set of states,
- Σ is the input alphabet,

- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states.

2.2 Week 2

Homework 1

Problem 1: Characterizing Accepted Sequences

The given problem involves designing a finite automaton that accepts sequences of 5 and 10-cent inputs summing to 25 cents.

Solution: We define the equation:

$$5a + 10b = 25 \tag{1}$$

where a is the number of 5-cent inputs and b is the number of 10-cent inputs. Solving for valid pairs:

- $(a = 5, b = 0) \Rightarrow$ Sequence: $5, 5, 5, 5, 5$
- $(a = 3, b = 1) \Rightarrow$ Sequence: $5, 5, 5, 10$
- $(a = 1, b = 2) \Rightarrow$ Sequence: $5, 10, 10$

These sequences are precisely those accepted by the automaton. The machine accepts a sequence if the total sum equals 25 cents.

Problem 2: Defining Valid Variable Names

A valid variable name must begin with a letter (ℓ) and be followed by any number of letters or digits (d).

Regular Expression:

$$\ell(\ell|d)^* \tag{2}$$

Solution: *Finite Automaton:* - States: q_0 (initial), q_1 (accepting). - Transitions: - $q_0 \to q_1$ on input ℓ - $q_1 \to q_1$ on input ℓ or d

Problem 3: Classification of Words in L_1, L_2, L_3

The given languages are defined as follows:

- $L_1 = \{x0y \mid x, y \in \Sigma^*\}$: The set of words that contain at least one '0'.
- $L_2 = \{w \mid |w| = 2^n \text{ for some } n \in \mathbb{N}\}$: The set of words whose length is a power of 2.
- $L_3 = \{w \mid |w|_0 = |w|_1\}$: The set of words where the number of 0s equals the number of 1s.

Solution: We analyze each word based on these conditions:

	L_1	L_2	L_3
$w_1 = 10011$	✓		
$w_2 = 100$	✓		
$w_3 = 10100100$	✓	✓	
$w_4 = 1010011100$	✓		✓
$w_5 = 11110000$	✓	✓	✓

Table 1: Classification of words into L_1, L_2, L_3

Problem 4: DFA Analysis

Given the DFA with states q_0 (start), q_2, and q_1 (accepting), we determine which words end in the accepting state q_1.

Transitions:

$$\begin{aligned} &\delta(q_0, 1) = q_0, && \delta(q_0, 0) = q_2 \\ &\delta(q_2, 0) = q_2, && \delta(q_2, 1) = q_1 \\ &\delta(q_1, 0) = q_1, && \delta(q_1, 1) = q_1 \end{aligned}$$

Checking Words:

- $w_1 = 0010$: $q_0 \to q_2 \to q_2 \to q_1 \to q_1$ ✓(Accepted)
- $w_2 = 1101$: $q_0 \to q_0 \to q_0 \to q_2 \to q_1$ ✓(Accepted)
- $w_3 = 1100$: $q_0 \to q_0 \to q_0 \to q_2 \to q_2$ (Not Accepted)

Solution:

$w_1 = 0010 \to$ ✓*Accepted*
$\mathrm{w}_2 = 1101 \to$ ✓*Accepted*
$\mathrm{w}_3 = 1100 \to$ *Rejected*

This confirms that w_1 and w_2 end in the accepting state, while w_3 does not.

Chapter 2.1 Report:

Chapter 2.1 discusses the use of finite automata in modeling real-world protocols, particularly in the context of electronic money transactions. The section introduces a three-party system involving a customer, a store, and a bank. The goal is to ensure that digital money is not duplicated or reused fraudulently.

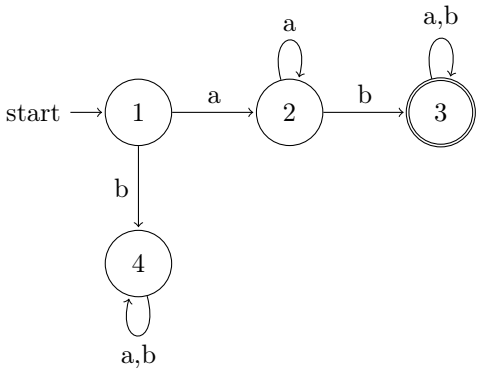
The protocol consists of five primary actions: pay, cancel, ship, redeem, and transfer. Each party's behavior is modeled using finite automata to track transaction states. The section highlights how such models can reveal vulnerabilities—such as a store shipping goods before verifying payment—showcasing the importance of automata in validating protocol security.

Finite automata prove to be useful for detecting logical flaws in transaction systems, ensuring valid sequences of operations. The chapter serves as an introduction to the application of formal computational models in the security and validation of protocols.

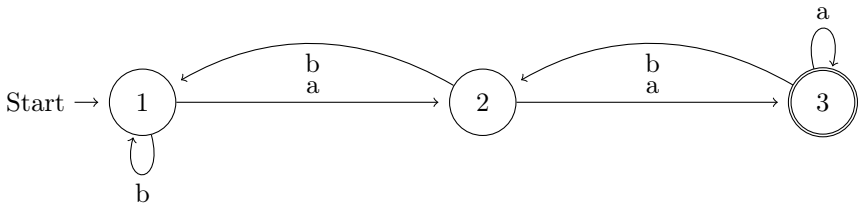
2.3 Week 3

Lecture Summary
TBD

Homework 2
Exercise 2: Implementing DFA Runs



DFA A_1



DFA A_2

Words accepted or refused by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively

w	Accepted A_1	Accepted A_2
aaa	×	✓
aab	✓	×
aba	×	×
abb	×	×
baa	×	✓
bab	×	×
bba	×	×
bbb	×	×

The table above summarizes the words accepted or rejected by DFA A_1 and DFA A_2. To implement these automata *programmatically*, we define the DFA class in `dfa.py`, which allows us to process input words according to their respective state transition diagrams.

DFA Implementation in `dfa.py`
This introduction describes the design of the `dfa.py`, consisting of:

- Q - a finite set of states.
- Σ - an input alphabet.
- $\delta : Q \times \Sigma \to Q$ - a transition function.
- $q_0 \in Q$ an initial state.
- $F \subseteq Q$ - a set of final accepting states.

The `DFA` class constructor takes these five elements (`Q`, `Sigma`, `delta`, `q0`, and `F`), using the method:

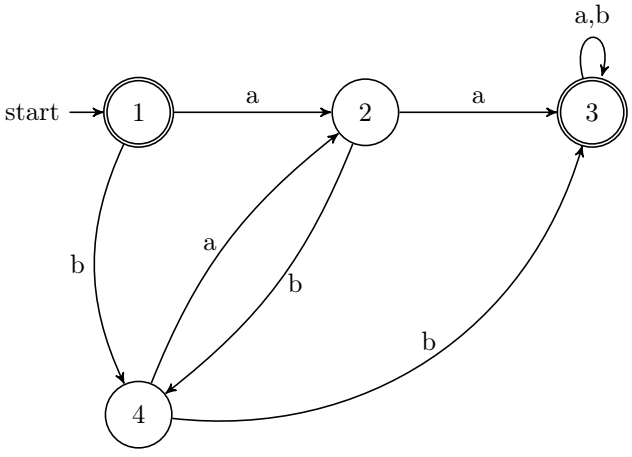
- `run(w)`: Runs the DFA on input string `w` and determines whether or not `w` is accepted based on the state it finishes at.

Implementation In the following code snippet, the `run` method processes the symbols of the input `w` sequentially, looking up the next state based on the current state and the input symbol. If an invalid transition is encountered, the method immediately returns `False`. Otherwise, if the DFA ends in a state that is a member of `F`, `True` is returned (meaning `w` is accepted); if it ends in some other state, `False` is returned.

```
class DFA :

    # init the DFA
    def __init__(self, Q, Sigma, delta, q0, F) :
        self.Q = Q # set of states
        self.Sigma = Sigma # set of symbols
        self.delta = delta # transition function
        self.q0 = q0 # initial state
        self.F = F # final states

   # print the data of the DFA
    def __repr__(self) :
        return f"DFA({self.Q},\n\t{self.Sigma},\n\t{self.delta},\n\t{self.q0},\n\t{self.F})"

    # run the DFA on the word w
    # return if the word is accepted or not
    # modify as needed
    def run(self, w) :
        # todo
        # start at initial state
        current_state = self.q0
        for symbol in w:
            if (current_state, symbol) in self.delta:
                current_state = self.delta[(current_state, symbol)]
            else:
                # invalid transition (dead state)
                return False
            # accept if in final state
        return current_state in self.F
```

Exercise 4: A new automaton from an old one

Below is DFA A_0 which accepts exactly the words that A refuses and vice versa.



In A_0, nodes 1 and 3 are the accepting final states, while nodes 2 and 4 are normal states.

Exercise 2.2.4: DFAs over $\{0, 1\}$

(a) The set of all strings ending in 00

DFA Description:

$$
\begin{aligned}
Q &= \{ q_0, q_1, q_2 \}, \\
\Sigma &= \{0, 1\}, \\
\delta &\text{ is given by the table below,} \\
q_0 &\text{ is the start state,} \\
F &= \{ q_2 \}.
\end{aligned}
$$

Transition Table:

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_0

Explanation:

- q_0 - we have not yet seen a trailing zero
- q_1 - the string currently ends in exactly one zero
- q_2 (accepting) - the string ends in at least two consecutive zeros

(b) The set of all strings with three consecutive 0s

DFA Description:

$$
\begin{aligned}
Q &= \{ q_0, q_1, q_2, q_3 \}, \\
\Sigma &= \{0, 1\}, \\
\delta &\text{ is given by the table below,} \\
q_0 &\text{ is the start state,} \\
F &= \{ q_3 \}.
\end{aligned}
$$

Transition Table:

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
q_3	q_3	q_3

Explanation:

- q_0 - we have seen 0 consecutive zeros so far
- q_1 - we have seen exactly 1 consecutive zero
- q_2 - we have seen exactly 2 consecutive zeros
- q_3 (accepting) - we have seen at least 3 consecutive zeros

(c) The set of all strings with `011` as a substring

DFA Description:

$$
\begin{aligned}
Q &= \{\, q_0,\, q_1,\, q_2,\, q_3\}, \\
\Sigma &= \{0, 1\}, \\
\delta &\text{ is given by the table below,} \\
q_0 &\text{ is the start state,} \\
F &= \{\, q_3\}.
\end{aligned}
$$

Transition Table:

δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_3	q_3

Explanation:

- q_0 - no partial match yet
- q_1 - we matched a single `0`
- q_2 - we matched `01`
- q_3 (accepting) - we found `011` somewhere in the string

2.4 Week 4

Lecture Summary
We explore the concept of *product automata* and how they can be used to model operations on languages accepted by two deterministic finite automata (DFAs). In particular, we discuss how to construct an automaton that accepts the intersection (or union) of the languages recognized by two given automata.

Homework 3

Problem 1: Extended transition function
Consider two DFAs:

$$\mathcal{A}^{(1)} = \big(Q^{(1)}, \Sigma, \delta^{(1)}, q_0^{(1)}, F^{(1)}\big) \quad \text{and} \quad \mathcal{A}^{(2)} = \big(Q^{(2)}, \Sigma, \delta^{(2)}, q_0^{(2)}, F^{(2)}\big),$$

over the alphabet $\Sigma = \{a, b\}$.

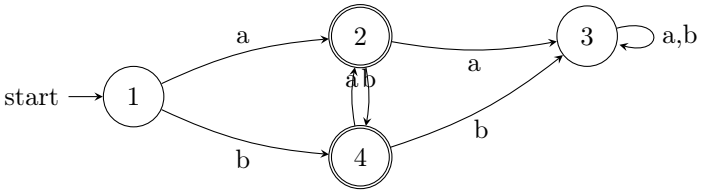


Diagram for $A^{(1)}$:

- State 1 is the initial state (arrow from the left).
- From state 1: reading 'a' goes to 2; reading 'b' goes to 4.

- From state 2: reading 'a' goes to 3; reading 'b' goes to 4.
- From state 4: reading 'a' goes to 2; reading 'b' goes to 3.
- From state 3: reading either 'a' or 'b' loops on 3.
- States 2 and 4 are accepting, while state 3 is a non-accepting sink.

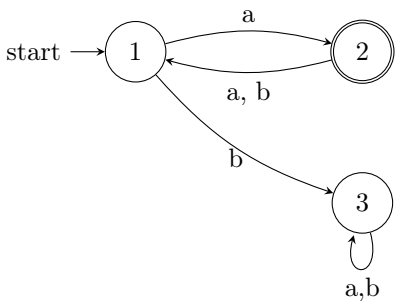


Diagram for $A^{(2)}$:

- State 1 is the initial state (arrow from the left).
- From state 1: reading 'a' goes to 2; reading 'b' goes to 3.
- From state 2: reading 'a' or 'b' goes back to 1.
- State 3 loops on both 'a' and 'b'.
- State 2 is the only accepting state.

a. Descriptions of Accepted Languages

- $\mathcal{A}^{(1)}$**:** Accepts non-empty words in which no two consecutive letters are the same.
- $\mathcal{A}^{(2)}$**:** Accepts words of odd length where every letter at an odd position of w is the letter a.

Hence,

$$L\big(A^{(2)}\big) = \big\{\, w \in \{a,b\}^* \mid |w| \text{ is odd and the odd-indexed letters are all } a \big\}.$$

b. Computation: $\widehat{\delta}^{(1)}\big(1, \texttt{abaa}\big)$

The automaton $A^{(1)}$ has states $\{1,2,3,4\}$, where:

- State 1 is the initial state (non-final).
- State 2 indicates the last letter read was a (no violation).
- State 4 indicates the last letter read was b (no violation).
- State 3 is the "sink" or "trap" state (once a violation occurs, e.g. two consecutive letters the same).

We compute step-by-step for the input `abaa`:

$$\widehat{\delta}^{(1)}(1, \texttt{abaa}) :$$

1. Start in state 1, input = `abaa`.
2. Read 'a': $\delta^{(1)}(1,a) = 2$. Remaining input = `baa`.
3. Now in state 2, read 'b': $\delta^{(1)}(2,b) = 4$. Remaining input = `aa`.

4. Now in state 4, read 'a': $\delta^{(1)}(4, a) = 2$. Remaining input = `a`.
5. Now in state 2, read 'a': $\delta^{(1)}(2, a) = 3$. Remaining input is empty.

Therefore,

$$\widehat{\delta}^{(1)}(1, \texttt{abaa}) \;=\; 3.$$

Since state 3 is the non-accepting sink, the string `abaa` is not accepted by $A^{(1)}$.

b. Computation: $\widehat{\delta}^{(2)}\big(1, \texttt{abba}\big)$

Automaton $A^{(2)}$ can be thought of as having states:

- State 1: an even number of letters read so far (initial, non-final).
- State 2: an odd number of letters read so far, with "odd positions must be a" still satisfied (accepting).
- State 3: dead/trap state (if the condition on odd positions is violated).

We compute for `abba` (positions are 1,2,3,4):

$$\widehat{\delta}^{(2)}(1, \texttt{abba}) :$$

1. Start in state 1, input = `abba`.
2. Read 'a' (position 1): $\delta^{(2)}(1, a) = 2$. Remaining input = `bba`.
3. Now in state 2, read 'b' (position 2): $\delta^{(2)}(2, b) = 1$. Remaining input = `ba`.
4. Now in state 1, read 'b' (position 3): $\delta^{(2)}(1, b) = 3$. Remaining input = `a`.
5. Now in state 3, read 'a': $\delta^{(2)}(3, a) = 3$. Remaining input is empty.

Hence,

$$\widehat{\delta}^{(2)}(1, \texttt{abba}) = 3,$$

and state 3 is non-accepting. Therefore, `abba` is not accepted by $A^{(2)}$.

Problem 2: Product automata

We now define a *product automaton* A from $A^{(1)}$ and $A^{(2)}$ in order to recognize

$$L\big(A^{(1)}\big) \;\cap\; L\big(A^{(2)}\big).$$

a. Construct the intersection automaton A

States. The state set of A is the Cartesian product

$$Q \;=\; Q^{(1)} \;\times\; Q^{(2)}.$$

If $Q^{(1)} = \{1, 2, 3, 4\}$ and $Q^{(2)} = \{1, 2, 3\}$, then

$$Q = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3), (4,1), (4,2), (4,3)\}.$$

Initial state. $(1, 1)$, since 1 is the initial state of $A^{(1)}$ and 1 is the initial state of $A^{(2)}$.

Transition function. For $(p, q) \in Q$ and $x \in \{a, b\}$,

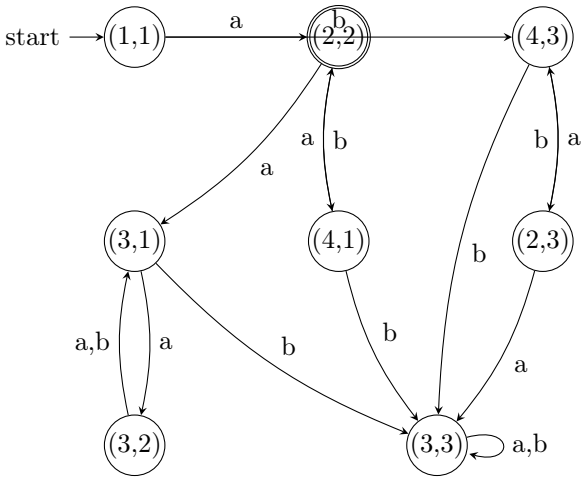
$$\delta\big((p, q), x\big) \;=\; \big(\delta^{(1)}(p, x),\; \delta^{(2)}(q, x)\big).$$

Final states (for intersection). A pair (p, q) is final if $p \in F^{(1)}$ and $q \in F^{(2)}$. - For $A^{(1)}$, we have $F^{(1)} = \{2, 4\}$ (all non-empty strings with no two consecutive letters the same). - For $A^{(2)}$, we have $F^{(2)} = \{2\}$ (odd-length strings with a in every odd position).

Thus

$$F \;=\; \{(p, q) \mid p \in \{2, 4\},\, q = 2\}.$$

Diagram of product automaton A showing all transitions. For brevity, not every state is drawn if some are unreachable.



b. Why $L(A) = L\big(A^{(1)}\big) \cap L\big(A^{(2)}\big)$

A string w is accepted by A precisely if:

- Following the transitions of A on w leads from $(1, 1)$ to a final state (p, q).
- This happens exactly when p is a final state of $A^{(1)}$ *and* q is a final state of $A^{(2)}$.

Thus w is accepted by both $A^{(1)}$ and $A^{(2)}$. Hence $L(A) = L(A^{(1)}) \cap L(A^{(2)})$.

c. Constructing A' for the Union

To get $L(A') = L\big(A^{(1)}\big) \cup L\big(A^{(2)}\big)$, we keep the same product structure but change the final states. A product state (p, q) is final if *either* $p \in F^{(1)}$ *or* $q \in F^{(2)}$. Formally:

$$F' \;=\; \big(F^{(1)} \times Q^{(2)}\big) \;\cup\; \big(Q^{(1)} \times F^{(2)}\big).$$

This ensures that a string is accepted by A' if it is accepted by $A^{(1)}$ *or* $A^{(2)}$.

Summary of Key Points:

- For $A^{(1)}$, the accepting states are $\{2, 4\}$; consecutive letters must differ and the string must be non-empty.
- For $A^{(2)}$, the accepting state is $\{2\}$; the string must have odd length and a in every odd position.
- Extended transitions showed that $\widehat{\delta}^{(1)}(1, \texttt{abaa}) = 3$ (not accepted) and $\widehat{\delta}^{(2)}(1, \texttt{abba}) = 3$ (not accepted).
- The product automaton for intersection has final states $F^{(1)} \times F^{(2)} = \{(2, 2), (4, 2)\}$, while for union we use $F' = (F^{(1)} \times Q^{(2)}) \cup (Q^{(1)} \times F^{(2)})$.

Exercise 2.2.7: Induction Proof

Statement: Let A be a DFA, and let q be a particular state of A such that

$$\delta(q, a) \;=\; q \quad \text{for all input symbols } a.$$

Show by induction on the length of the input that for all input strings w, we have

$$\widehat{\delta}(q, w) \;=\; q.$$

Proof (by induction on the length of w):

Base Case ($|w| = 0$): If w is the empty string ϵ, then by definition of the extended transition function,

$$\widehat{\delta}(q, \epsilon) \;=\; q.$$

Hence the statement holds for $|w| = 0$.

Inductive Step: Assume that for all strings x of length n, we have

$$\widehat{\delta}(q, x) \;=\; q.$$

We need to prove the statement for any string w of length $n+1$. Let w be such a string. We can write w as

$$w \;=\; x\,a,$$

where x is a string of length n, and a is a single input symbol. Then,

$$\widehat{\delta}(q, w) \;=\; \widehat{\delta}(q,\, x\,a) \;=\; \delta\big(\widehat{\delta}(q, x),\, a\big).$$

By the inductive hypothesis, $\widehat{\delta}(q, x) = q$. Therefore,

$$\widehat{\delta}(q, w) \;=\; \delta(q, a).$$

But we are given that $\delta(q, a) = q$ for all symbols a. Hence,

$$\widehat{\delta}(q, w) \;=\; q.$$

This completes the inductive step.

Conclusion: By the principle of mathematical induction, for all strings w, we have

$$\widehat{\delta}(q, w) = q.$$

Thus, if a state q transitions to itself on every symbol, it remains q under any input string.

ITALC 2.3 Question: In the subset construction for converting an NFA to a DFA, often there is an exponential blow-up in the number of states. Can you propose a scenario for which this blow-up actually happens, and whether there are any strategies or special cases that might mitigate this worst-case behavior?

2.5 Week 5

Lecture Summary
TBD

Homework 4
Problem 1: NFA Interpretation We can consider A as an NFA because each transition goes to a single-element set (instead of exactly one state). Hence, A can be viewed as a valid NFA.

$$A' = \big(Q', \Sigma, \delta', q_0', F'\big),$$

where

$$Q' = Q, \quad q_0' = q_0, \quad F' = F, \quad \delta'(q, a) = \{\delta(q, a)\}.$$

2. Why it works.

Each string accepted by A has the exact same single path of states in A', because δ' uses the same transitions as δ, but returns them as singleton sets. Thus, every string accepted by A' follows that same single path of states in A. Consequently,

$$L(A') \;=\; L(A).$$

Problem 2: The NFA A has four states q_0, q_1, q_2, q_3, with q_0 as the start state and q_3 as the (only) accepting state. The transitions are:

$$\delta(q_0, 0) = \{q_1\}, \quad \delta(q_0, 1) = \{q_1\},$$
$$\delta(q_1, 0) = \{q_2\}, \quad \delta(q_1, 1) = \{q_2, q_3\},$$
$$\delta(q_2, 0) = \varnothing, \quad \delta(q_2, 1) = \{q_3\},$$
$$\delta(q_3, 0) = \varnothing, \quad \delta(q_3, 1) = \varnothing.$$

There are no loops, so only strings of length 2 or 3 can reach q_3. Specifically:

- For length 2:
$$q_0 \xrightarrow{\text{0 or 1}} q_1 \xrightarrow{1} q_3,$$
which corresponds to the strings {01, 11}.
- For length 3:
$$q_0 \xrightarrow{\text{0 or 1}} q_1 \xrightarrow{\text{0 or 1}} q_2 \xrightarrow{1} q_3,$$
which corresponds to the strings {001, 011, 101, 111}.

Hence, the language is:

$$L(A) \;=\; \{\,01,\, 11,\, 001,\, 011,\, 101,\, 111\}.$$

2. Specify A in the form $(Q, \Sigma, \delta, q_0, F)$

$$Q = \{q_0,\, q_1,\, q_2,\, q_3\}, \quad \Sigma = \{0, 1\}, \quad q_0 \text{ is the start state}, \quad F = \{q_3\}.$$

The transition function δ is as above.

3. Compute $\hat{\delta}\big(q_0,\, 10110\big)$ step by step

We track the set of possible states after each input symbol:

1. Start: $\{q_0\}$.
2. Read '1': $\delta(q_0, 1) = \{q_1\}$, so now in $\{q_1\}$.

3. Read '0': $\delta(q_1, 0) = \{q_2\}$, so now in $\{q_2\}$.
4. Read '1': $\delta(q_2, 1) = \{q_3\}$, so now in $\{q_3\}$.
5. Read '1': $\delta(q_3, 1) = \varnothing$, so now in $\varnothing$.
6. Read '0': from $\varnothing$, we stay in $\varnothing$.

Thus,

$$\hat{\delta}(q_0,\, 10110) \;=\; \varnothing.$$

Since $\varnothing$ does not contain a final state, 10110 is not accepted.

4. Find all paths in A for $v = 1$ and $w = 1010$

For $v = 1$:

$$q_0 \xrightarrow{1} q_1.$$

No more input; we end in q_1. Since $q_1 \notin F$, the string 1 is not accepted.

For $w = 1010$:

$$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} \varnothing.$$

We end in $\varnothing$, so 1010 is not accepted. In both cases, there is a unique path because the transitions are forced by the input symbols.

5. Construct the determinization A^D (power-set construction)

We list the reachable subsets of $\{q_0, q_1, q_2, q_3\}$:

- **Start state**: $\{q_0\}$.
  - On 0 or 1, go to $\{q_1\}$.
- **State** $\{q_1\}$.
  - On 0: $\{q_2\}$.
  - On 1: $\{q_2, q_3\}$.
- **State** $\{q_2\}$.
  - On 0: $\varnothing$.
  - On 1: $\{q_3\}$.
- **State** $\{q_2, q_3\}$.
  - On 0: $\varnothing$.
  - On 1: $\{q_3\}$.
- **State** $\{q_3\}$.
  - On 0: $\varnothing$.
  - On 1: $\varnothing$.
- **Dead state** $\varnothing$.
  - On 0 or 1: $\varnothing$.

The start state in the DFA is $\{q_0\}$, and the accepting states are all subsets containing q_3, namely $\{q_2, q_3\}$ and $\{q_3\}$.

6. Verify $L(A) = L(A^D)$. Is there a smaller DFA?

By the standard subset construction, we have $L(A^D) = L(A)$. Since $L(A)$ is the finite set of strings {01, 11, 001, 011, 101, 111}, one might suspect a smaller DFA could exist. However, standard DFA minimization shows that all the reachable states in A^D are pairwise inequivalent, so no merges are possible without altering the language. Hence, the 6-state DFA is already minimal.

Question: Can you describe a practical scenario where combining the two (i.e first designing an NFA and then converting or integrating it into a DFA) provides benefits that neither approach alone can normally offer?

2.6 Week 6

Lecture Summary
TBD

Homework 5
Exercise 3.2.1: Determinization of an NFA

Consider an NFA

$$N = (Q, \Sigma, \delta, q_0, F),$$

where

- Q is the set of states,
- Σ is the alphabet,
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function,
- q_0 is the initial state, and
- $F \subseteq Q$ is the set of final (accepting) states.

The subset construction (or powerset construction) produces a DFA

$$D = (Q', \Sigma, \delta', q_0', F'),$$

where:

- $q_0' = \{q_0\}$ (or the ϵ-closure of q_0 if ϵ-transitions are present),
- Q' is the set of all subsets of Q that are *reachable* from $\{q_0\}$ using the rule

  $$\delta'(S, a) = \bigcup_{q \in S} \delta(q, a)$$

  for each $S \subseteq Q$ and $a \in \Sigma$,
- $F' = \{S \in Q' \mid S \cap F \neq \emptyset\}$.

a. Description of the Algorithm

1. Initialize the DFA with the start state $S_0 = \{q_0\}$.
2. For each state S in Q' (initially, just S_0) and for each symbol $a \in \Sigma$, compute

   $$\delta'(S, a) = \bigcup_{q \in S} \delta(q, a).$$

   If this subset has not been seen before, add it to Q'.

3. Continue until no new subsets are generated.

b. Proof of Equivalence

Soundness: Suppose the DFA D accepts a word w. Then the state S reached after processing w satisfies $S \cap F \neq \emptyset$. By construction, S represents all possible states in N after reading w; hence, there is at least one computation in N that reaches an accepting state.

Completeness: Conversely, if N accepts w then some computation ends in a state $q \in F$. Consequently, the subset S reached in D contains q (i.e., $S \cap F \neq \emptyset$), and thus D accepts w.

c. Example Application

Let

$$Q = \{q_0, q_1, q_2\}, \quad \Sigma = \{0, 1\}, \quad q_0 \text{ is the start state}, \quad F = \{q_2\},$$

with transitions:

$$\delta(q_0, 0) = \{q_0, q_1\}, \quad \delta(q_0, 1) = \{q_0\}, \quad \delta(q_1, 1) = \{q_2\},$$

and all other transitions mapping to $\emptyset$.

The subset construction proceeds as follows:

- Start with $S_0 = \{q_0\}$.
- From S_0:
$$\delta'(S_0, 0) = \{q_0, q_1\} \triangleq S_1, \quad \delta'(S_0, 1) = \{q_0\} = S_0.$$
- From $S_1 = \{q_0, q_1\}$:
$$\delta'(S_1, 0) = \{q_0, q_1\} = S_1, \quad \delta'(S_1, 1) = \{q_0, q_2\} \triangleq S_2.$$
- From $S_2 = \{q_0, q_2\}$:
$$\delta'(S_2, 0) = \{q_0, q_1\} = S_1, \quad \delta'(S_2, 1) = \{q_0\} = S_0.$$

Thus, the DFA has states $S_0 = \{q_0\}$, $S_1 = \{q_0, q_1\}$, and $S_2 = \{q_0, q_2\}$ with S_2 as the sole accepting state.

d. Observations

Even though the NFA has 3 states, the worst-case DFA could have up to $2^3 = 8$ states. In this example, only 3 reachable subsets occur, demonstrating that the algorithm generates only the necessary states.

Exercise 3.2.2: Analysis of a Specific NFA

Consider the NFA

$$N = (Q, \Sigma, \delta, q_0, F)$$

with:

- $Q = \{q_0, q_1, q_2, q_3\}$,
- $\Sigma = \{a, b\}$,
- q_0 is the start state,
- $F = \{q_3\}$, and
- The transition function is given by:

$$\begin{aligned} \delta(q_0, a) &= \{q_0, q_1\}, \\ \delta(q_0, b) &= \{q_0\}, \\ \delta(q_1, b) &= \{q_2\}, \\ \delta(q_2, a) &= \{q_3\}. \end{aligned}$$

a. Reachable Subsets

Starting with $S_0 = \{q_0\}$:

- $\delta'(S_0, a) = \{q_0, q_1\} \triangleq S_1$,
- $\delta'(S_0, b) = \{q_0\} = S_0$.

From $S_1 = \{q_0, q_1\}$:

- $\delta'(S_1, a) = \{q_0, q_1\} = S_1$,
- $\delta'(S_1, b) = \{q_0, q_2\} \triangleq S_2$.

From $S_2 = \{q_0, q_2\}$:

- $\delta'(S_2, a) = \{q_0, q_1, q_3\} \triangleq S_3$,
- $\delta'(S_2, b) = \{q_0\} = S_0$.

From $S_3 = \{q_0, q_1, q_3\}$:

- $\delta'(S_3, a) = \{q_0, q_1\} = S_1$,
- $\delta'(S_3, b) = \{q_0, q_2\} = S_2$.

Thus, the reachable subsets are:

$$S_0 = \{q_0\}, \quad S_1 = \{q_0, q_1\}, \quad S_2 = \{q_0, q_2\}, \quad S_3 = \{q_0, q_1, q_3\}.$$

b. DFA Transition Diagram

The DFA constructed has:

- **States:** S_0, S_1, S_2, S_3,
- **Start state:** S_0,
- **Accepting state:** S_3 (since $q_3 \in S_3$).

The transitions are:

$$\begin{aligned}
\delta'(S_0, a) &= S_1, \quad \delta'(S_0, b) = S_0, \\
\delta'(S_1, a) &= S_1, \quad \delta'(S_1, b) = S_2, \\
\delta'(S_2, a) &= S_3, \quad \delta'(S_2, b) = S_0, \\
\delta'(S_3, a) &= S_1, \quad \delta'(S_3, b) = S_2.
\end{aligned}$$

A state diagram can be drawn with these transitions.

c. Proof that No Additional States are Reachable

Every new subset is generated by applying δ' to an existing subset. In this example, starting from S_0 only S_1, S_2, and S_3 are generated. No transition yields a subset outside $\{S_0, S_1, S_2, S_3\}$. Thus, these four subsets are the complete set of reachable states.

d. Observations

Although the original NFA has 4 states (and a worst-case DFA might have $2^4 = 16$ states), only 4 reachable subsets occur. This illustrates that the nondeterminism of the NFA does not necessarily lead to the full exponential blowup in the number of states in practice.

Exercise 4.4.1: DFA Minimization via Distinguishability
Consider the DFA with states where the start state is A and the only final state is D. The transition function is given in Table 2.

$$\{A, B, C, D, E, F, G\},$$

State	0	1
A	B	A
B	A	C
C	B	D
$*D$	E	F
E	F	G
F	G	D
G	F	D

Table 2: Transition function for the DFA in Exercise 4.4.1. Here, state D (marked with an asterisk) is final.

a. Table of Distinguishability

We first mark every pair (p, q) where exactly one of p and q is final. Since D is the only final state, all pairs of the form (D, X) with $X \neq D$ are marked.

Then, using the standard table-filling algorithm, we iteratively mark pairs (p, q) if there exists an input symbol a such that the pair of transitions $(\delta(p, a), \delta(q, a))$ is already marked. After processing all pairs, we find that every pair of distinct states is marked except for the pair (F, G). This indicates that states F and G are *indistinguishable* (i.e., equivalent).

b. The Minimum-State Equivalent DFA

By merging the equivalent states F and G into a single state, denoted by $[FG]$, the minimized DFA has the following states:

$$\{[A], [B], [C], [D], [E], [FG]\}.$$

The transition function for the minimized DFA is as follows:

- $[A]: \quad 0 \to [B], \quad 1 \to [A],$
- $[B]: \quad 0 \to [A], \quad 1 \to [C],$
- $[C]: \quad 0 \to [B], \quad 1 \to [D],$
- $[D]: \quad 0 \to [E], \quad 1 \to [FG],$
- $[E]: \quad 0 \to [FG], \quad 1 \to [FG],$
- $[FG]: \quad 0 \to [FG], \quad 1 \to [D].$

Thus, the minimized DFA contains 6 states.

Exercise 4.4.2: Minimization of a 9-State DFA

Consider the DFA with states

$$\{A, B, C, D, E, F, G, H, I\},$$

with A as the start state and C and I as final states. Its transitions are given in Table 3.

a. Table of Distinguishability

Again, we begin by marking all pairs in which one state is final and the other is nonfinal. Thus, every pair containing exactly one of C or I is marked immediately.

Next, we consider the pairs of nonfinal states $\{A, B, D, E, F, G, H\}$. By applying the table-filling algorithm (i.e., if for some input symbol the transitions from the pair lead to a marked pair, then mark the original pair), we eventually mark every pair among these states.

State	0	1	Final?
A	B	E	No
B	C	F	No
$*C$	D	G	Yes
D	E	H	No
E	F	I	No
F	G	B	No
G	H	B	No
H	I	B	No
$*I$	H	E	Yes

Table 3: Transition function for the DFA in Exercise 4.4.2. States C and I are final.

Finally, we check the pair (C, I) of final states. For example, on input 0, we have $\delta(C, 0) = D$ and $\delta(I, 0) = H$. The pair (D, H) is marked because, for some input, it leads to distinguishable outcomes. Thus, (C, I) is also marked.

Hence, every pair of distinct states is distinguishable.

b. The Minimum-State DFA

Since all pairs of distinct states are distinguishable, no states can be merged. Therefore, the original DFA is already minimal and consists of 9 states:

$$\{A, B, C, D, E, F, G, H, I\}.$$

ITALC Question: In practice, is it always worth fully minimizing a DFA when implementing features like pattern matching or lexical analysis?

2.7 Week 7

Lecture Summary
TBD

Homework 6
Exercise A:

1. **Input language:** $L = \{\, 1\,0^n : n \in \mathbb{N}\}$. on input $1\,0^n$ the TM must output $1\,0^{n+1}$.

$$M_1 = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

$$Q = \{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\},\ \Sigma = \{0, 1\},\ \Gamma = \{0, 1, B\},\ F = \{q_{\text{accept}}\}.$$

	0	1	B
q_0	$(q_{\text{reject}}, 0, R)$	$(q_1, 1, R)$	–
q_1	$(q_1, 0, R)$	$(q_{\text{reject}}, 1, R)$	$(q_{\text{accept}}, 0, R)$
q_{accept}		halt+accept	
q_{reject}		halt+reject	

2. **Input language:** $L = \{\, 1\,0^n : n \in \mathbb{N}\}$. on input $1\,0^n$ the TM must leave just the leading 1.

$$M_2 = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

$Q = \{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\}$.

	0	1	B
q_0	$(q_{\text{reject}}, 0, R)$	$(q_1, 1, R)$	–
q_1	(q_1, B, R)	$(q_{\text{reject}}, 1, R)$	$(q_{\text{accept}}, B, R)$
q_{accept}		halt+accept	
q_{reject}		halt+reject	

3. **Input language:** all binary strings. the TM outputs the bitwise complement (swap $0 \leftrightarrow 1$).

$$M_3 = (Q, \Sigma, \Gamma, \delta, q_0, B, F),\ Q = \{q_0, q_{\text{accept}}\}$$

	0	1	B
q_0	$(q_0, 1, R)$	$(q_0, 0, R)$	$(q_{\text{accept}}, B, R)$
q_{accept}		halt+accept	

Question: If we encode the step bound k in unary (1^k) instead of binary, does the language $L_3 = \{\langle M, w, k\rangle \mid M$ halts on w in $\leq k$ steps$\}$ remain decidable, or does the change in encoding alter its classification?

2.8 Week 8

Lecture Summary
TBD

Homework 7
Exercise 1: Decidability
Classify each language as *decidable*, *recursively enumerable (r.e.)*, or *co-r.e.*.

1. $L_1 = \{\, M \mid M$ halts on itself$\}$
   - **r.e. but not decidable**; not co-r.e.
   - Argument: Simulate M on its own description; accept if it halts (semi-decider). Decidable would imply a solution to the halting problem.
2. $L_2 = \{\, \langle M, w\rangle \mid M$ halts on $w\}$
   - **r.e. but not decidable**; not co-r.e.
   - Classic halting problem.
3. $L_3 = \{\, \langle M, w, k\rangle \mid M$ halts on w in $\leq k$ steps$\}$
   - **Decidable** (thus both r.e. and co-r.e.).
   - Simulate exactly k steps, accept if halts; otherwise reject.

Exercise 2: Closure Properties
For each statement, indicate True/False and give justification.

1. L_1, L_2 decidable $\Rightarrow L_1 \cup L_2$ decidable. **True**. Run both deciders; accept if either accepts.
2. L decidable $\Rightarrow \overline{L}$ decidable. **True**. Flip the accept/reject outcome.
3. L decidable $\Rightarrow L^*$ decidable. **True**. Enumerate all segmentations; dynamic-programming decider halts.

4. L_1, L_2 r.e. $\Rightarrow L_1 \cup L_2$ r.e. **True**. Dovetail the two semi-deciders.
5. L r.e. $\Rightarrow \overline{L}$ r.e. **False**. Counterexample: the halting problem; complement not r.e.
6. L r.e. $\Rightarrow L^*$ r.e. **True**. Enumerate finite concatenations of strings from an enumerator for L.

Question: For an r.e. language L we know that L^* is r.e.; what about the intersection $L \cap \overline{L}$? is it always decidable, and how does this relate to the closure results we discussed?

2.9 Week 9

Lecture Summary
Unless stated otherwise, every variable ranges over the naturals $\mathbb{N} = \{1, 2, 3, \dots\}$ and log denotes the natural logarithm.

Homework 8-9
Exercise 1: Growth order Order the functions from *slowest* to *fastest* growth:

$$\log(\log n),\ \log n,\ e^{\log n},\ e^{2\log n},\ 2^n,\ e^n,\ n!,\ 2^{2^n}.$$

1. $\log(\log n)$ $\lim_{n\to\infty} \dfrac{\log(\log n)}{\log n} = 0$ by L'Hospital, so $\log(\log n) = o(\log n)$.
2. $\log n$ For every $\varepsilon > 0$, $\lim_{n\to\infty} \dfrac{\log n}{n^\varepsilon} = 0$, hence $\log n = o(n^\varepsilon)$ and is beaten by any polynomial.
3. $e^{\log n} = n$ (inverse property of log/exp).
4. $e^{2\log n} = (e^{\log n})^2 = n^2$.
5. 2^n $\lim_{n\to\infty} \dfrac{n^k}{2^n} = 0$ for every fixed k, so every polynomial is $o(2^n)$.
6. e^n $2^n = e^{n \log 2}$; since $\log 2$ is a constant, $2^n \in \Theta(e^n)$ (same class, different base).
7. $n!$ Stirling: $n! \sim \sqrt{2\pi n}\,(n/e)^n \gg a^n$ for any fixed $a > 1$.
8. 2^{2^n} Double exponential: for all $a > 1$, $a^n = o(2^{2^n})$.

Exercise 2: Basic properties of $\mathcal{O}$ Definition used: $f \in \mathcal{O}(g) \iff \exists M > 0,\ N\ \forall n \geq N :\ f(n) \leq M\, g(n)$.

Let $f, g, h : \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $c > 0$.

1. $f \in \mathcal{O}(f)$: choose $M = 1$, $N = 1$.
2. $\mathcal{O}(cf) = \mathcal{O}(f)$:

   Forward — if $u \leq M\, c\, f$ for $n \geq N$, set $M' = Mc$. *Reverse* — if $u \leq M\, f$, then $u \leq M\, cf$ since $c > 0$.
3. If $f(n) \leq g(n)$ for $n \geq N_0$ then $\mathcal{O}(f) \subseteq \mathcal{O}(g)$. Given $u \leq M_1 f$ for $n \geq N_1$, combine with $f \leq g$ on $n \geq N = \max\{N_0, N_1\}$ to get $u \leq M_1\, g$.
4. If $\mathcal{O}(f) \subseteq \mathcal{O}(g)$ then $\mathcal{O}(f + h) \subseteq \mathcal{O}(g + h)$. Because $f + h \in \mathcal{O}(f)$ and $\mathcal{O}(f) \subseteq \mathcal{O}(g)$, $\exists M_2, N_2$ with $f + h \leq M_2(g + h)$. If $u \leq M_1(f + h)$ for $n \geq N_1$, then $u \leq M_1 M_2\,(g + h)$ for $n \geq \max\{N_1, N_2\}$.
5. If $h(n) > 0$ and $\mathcal{O}(f) \subseteq \mathcal{O}(g)$ then $\mathcal{O}(fh) \subseteq \mathcal{O}(gh)$: multiply the inequality from part (3) by the positive $h(n)$.

Exercise 3: Consequences for common families Let $i, j, k \in \mathbb{N}$.

1. If $j \leq k$, then $n^j \leq n^k$ for $n \geq 1 \ \Rightarrow\ \mathcal{O}(n^j) \subseteq \mathcal{O}(n^k)$ (constants $M = 1, N = 1$).

2. Same hypothesis gives $n^j + n^k \le 2n^k$ for $n \ge 1$, so $\mathcal{O}(n^j + n^k) \subseteq \mathcal{O}(n^k)$ (multiply old M by 2).
3. For the polynomial $p(n) = \sum_{i=0}^{k} a_i n^i$ let $A := \sum_{i=0}^{k} |a_i|$. Then $\big|p(n)\big| \le A\, n^k$ for all $n \ge 1$, hence $p \in \mathcal{O}(n^k)$ with $M = A, N = 1$.
4. $\log n \le n$ for $n \ge 1 \Rightarrow \mathcal{O}(\log n) \subseteq \mathcal{O}(n)$.
5. For $n \ge 2$, $\log n \le n \Rightarrow n \log n \le n^2 \Rightarrow \mathcal{O}(n \log n) \subseteq \mathcal{O}(n^2)$.

Exercise 4: Comparing classes Use limits to prove containments.

1. $\mathcal{O}(n)$ vs. $\mathcal{O}(\sqrt{n})$,
2. $\mathcal{O}(n^2)$ vs. $\mathcal{O}(2^n)$,
3. $\mathcal{O}(\log n)$ vs. $\mathcal{O}((\log n)^2)$,
4. $\mathcal{O}(2^n)$ vs. $\mathcal{O}(3^n)$,
5. $\mathcal{O}(\log_2 n)$ vs. $\mathcal{O}(\log_3 n)$.

1. $\lim\limits_{n\to\infty} \dfrac{\sqrt{n}}{n} = 0 \Rightarrow \mathcal{O}(\sqrt{n}) \subsetneq \mathcal{O}(n)$.
2. $\lim\limits_{n\to\infty} \dfrac{n^2}{2^n} = 0 \Rightarrow \mathcal{O}(n^2) \subsetneq \mathcal{O}(2^n)$.
3. $\lim\limits_{n\to\infty} \dfrac{\log n}{(\log n)^2} = 0 \Rightarrow \mathcal{O}(\log n) \subsetneq \mathcal{O}((\log n)^2)$.
4. $\lim\limits_{n\to\infty} \dfrac{2^n}{3^n} = (2/3)^n = 0 \Rightarrow \mathcal{O}(2^n) \subsetneq \mathcal{O}(3^n)$.
5. $\log_2 n = \dfrac{\log_3 n}{\log_3 2}$ (constant factor) $\Rightarrow \mathcal{O}(\log_2 n) = \mathcal{O}(\log_3 n)$.

Exercise 5: Sorting algorithms Give worst-case complexity and a rationale.

1. *Bubble sort* and *Insertion sort*: both perform $\Theta(n^2)$ comparisons and swaps in the worst case (inner loop scans almost the whole array). Insertion sort usually beats bubble sort on nearly-sorted data because its inner while-loop stops early once the insertion point is found.
2. *Insertion sort* vs. *Merge sort*: insertion sort is $\Theta(n^2)$. Merge sort solves the recurrence $T(n) = 2T(n/2) + \Theta(n)$, giving $T(n) = \Theta(n \log n)$ by the Master Theorem, thus asymptotically faster.
3. *Merge sort* vs. *Quick sort*: merge sort is $\Theta(n \log n)$ in *all* cases. Quick sort is $\Theta(n \log n)$ on average (good pivots) but $\Theta(n^2)$ in the worst case (already-sorted input with naive pivots). Therefore merge sort has better worst-case guarantees, while quick sort is often faster in practice due to low constants and cache-friendly in-place partitioning.

Questions: 1. Are there any *real* problems whose best algorithm runs in about $n^{\log n}$ time sitting between polynomial and exponential or is that gap still empty? 2. If I encode a graph super-bloated (unary weights, lots of repeats), does Kruskal suddenly stop being "poly-time" on a Turing machine? How much can encoding choice mess with our "this problem is in P" claims?

2.10 Week 10

Lecture Summary
TBD

Homework 10–11

Exercise 1: Converting formulas to CNF

Rewrite each formula in conjunctive normal form (CNF). Show every logical equivalence used.

1. $\varphi_1 := \neg((a \wedge b) \vee (\neg c \wedge d))$

   ‘‘‘

   $$\begin{aligned} \varphi_1 &\equiv \neg(a \wedge b) \wedge \neg(\neg c \wedge d) && \text{De Morgan} \\ &\equiv (\neg a \vee \neg b) \wedge (c \vee \neg d). \end{aligned}$$

   Hence the CNF consists of the two clauses $(\neg a \vee \neg b)$ and $(c \vee \neg d)$. ‘‘‘

2. $\varphi_2 := \neg\big((p \vee q) \to (r \wedge \neg s)\big)$

   ‘‘‘ First eliminate the implication:

   $$(p \vee q) \to (r \wedge \neg s) \;\equiv\; \neg(p \vee q) \vee (r \wedge \neg s).$$

   Then apply De Morgan again:

   $$\begin{aligned} \varphi_2 &\equiv \neg\big(\neg(p \vee q) \vee (r \wedge \neg s)\big) \\ &\equiv (p \vee q) \wedge \neg(r \wedge \neg s) \\ &\equiv (p \vee q) \wedge (\neg r \vee s). \end{aligned}$$

   The resulting CNF has the clauses $(p \vee q)$ and $(\neg r \vee s)$. ‘‘‘

Exercise 2: Satisfiability analysis

For each formula decide whether it is satisfiable. If it is, provide a satisfying assignment; otherwise give a short proof of impossibility.

1. $\psi_1 := (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$.

   Truth-table inspection shows the *only* model is $a = \mathsf{F},\ b = \mathsf{F}$, so ψ_1 is *satisfiable*.

2. $\psi_2 := (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg(\neg p \vee r)$.

   c

p	q	r	$\neg p \vee q$	$\neg q \vee r$	$\neg(\neg p \vee r)$	ψ_2
T	T	T	T	T	F	F
T	T	F	T	F	T	F
T	F	T	F	T	F	F
T	F	F	F	T	T	F
F	T	T	T	T	F	F
F	T	F	T	F	F	F
F	F	T	T	T	F	F
F	F	F	T	T	F	F

   Every row falsifies at least one conjunct, hence ψ_2 is *unsatisfiable*.

3. $\psi_3 := (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$.

   From the first two clauses we derive y; from the last two we derive $\neg y$. The contradictory requirements make ψ_3 *unsatisfiable*.

Exercise 3: Encoding *Sudoku* in CNF

Introduce propositional variables $x_{r,c,v}$ for rows $r \in \{1,\dots,9\}$, columns $c \in \{1,\dots,9\}$ and values $v \in \{1,\dots,9\}$, where $x_{r,c,v}$ is *true* iff entry (r,c) carries the number v.

The global constraint $\varphi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ is expressed by the following six CNFs.

C_1 **(*at least one value per cell*).**

$$C_1 \;=\; \bigwedge_{r,c} \Big(\bigvee_{v=1}^{9} x_{r,c,v} \Big).$$

C_2 **(*at most one value per cell*).**

$$C_2 \;=\; \bigwedge_{r,c} \bigwedge_{1 \le v < w \le 9} (\neg x_{r,c,v} \;\vee\; \neg x_{r,c,w}).$$

C_3 **(*each row contains every digit*).**

$$C_3 \;=\; \bigwedge_{r=1}^{9} \bigwedge_{v=1}^{9} \Big(\bigvee_{c=1}^{9} x_{r,c,v} \Big).$$

C_4 **(*each column contains every digit*).**

$$C_4 \;=\; \bigwedge_{c=1}^{9} \bigwedge_{v=1}^{9} \Big(\bigvee_{r=1}^{9} x_{r,c,v} \Big).$$

C_5 **(*each* 3×3 *block contains every digit*).**

$$C_5 \;=\; \bigwedge_{b_r=0}^{2} \bigwedge_{b_c=0}^{2} \bigwedge_{v=1}^{9} \Big(\bigvee_{r=3b_r+1}^{3b_r+3} \bigvee_{c=3b_c+1}^{3b_c+3} x_{r,c,v} \Big).$$

C_6 **(*given clues*).** For every pre-filled cell (r,c) with value v_0 add the unit clause x_{r,c,v_0}.

Bonus: Generic $n = k^2$ Sudoku.

Let $n = k^2$ with $k \ge 2$ and keep the variables $x_{r,c,v}$ for $1 \le r, c, v \le n$.

C_1 $\displaystyle\bigwedge_{r=1}^{n} \bigwedge_{c=1}^{n} \Big(\bigvee_{v=1}^{n} x_{r,c,v} \Big)$

C_2 $\displaystyle\bigwedge_{r=1}^{n} \bigwedge_{c=1}^{n} \bigwedge_{1 \le v < w \le n} (\neg x_{r,c,v} \vee \neg x_{r,c,w})$

C_3 $\displaystyle\bigwedge_{r=1}^{n} \bigwedge_{v=1}^{n} \Big(\bigvee_{c=1}^{n} x_{r,c,v} \Big)$

C_4 $\displaystyle\bigwedge_{c=1}^{n} \bigwedge_{v=1}^{n} \Big(\bigvee_{r=1}^{n} x_{r,c,v} \Big)$

C_5 $\displaystyle\bigwedge_{B_r=0}^{k-1} \bigwedge_{B_c=0}^{k-1} \bigwedge_{v=1}^{n} \Big(\bigvee_{r=kB_r+1}^{kB_r+k} \bigvee_{c=kB_c+1}^{kB_c+k} x_{r,c,v} \Big)$

C_6 For each given clue $(r, c) = v_0$ add the unit clause x_{r,c,v_0}.

These six families of clauses constitute a CNF whose models are in one-to-one correspondence with valid $k^2 \times k^2$ Sudoku grids.

Questions: 1. What is the smallest clause-count known for a complete Sudoku CNF that still lets modern SAT solvers finish each standard puzzle in under a second?