Languages and Algorithms for Artificial Intelligence (Module 3)

Last update: 28 March 2024

Academic Year 2023 – 2024

Alma Mater Studiorum · University of Bologna

Contents

1 Introduction

Computational task Description of a problem. Computational task

Computational process Algorithm to solve a task. Computational process

Algorithm (informal) A finite description of elementary and deterministic computation steps.

1.1 Notations

Set of the first n natural numbers Given $n \in \mathbb{N}$, we have that $[n] = \{1, \dots, n\}$.

1.1.1 Strings

Alphabet Finite set of symbols. Alphabet

String Finite, ordered, and possibly empty tuple of elements of an alphabet. String

The empty string is denoted as ε.

Strings of given length Given an alphabet S and $n \in \mathbb{N}$, we denote with S^n the set of all the strings over S of length n.

Kleene star Given an alphabet S, we denote with $S^* = \bigcup_{n=0}^{\infty} S^n$ the set of all the strings over S. Kleene star

Language Given an alphabet S, a language $\mathcal{L}$ is a subset of S^*. Language

1.1.2 Tasks encoding

Encoding Given a set A, any element $x \in A$ can be encoded into a string of the language $\{0, 1\}^*$. The encoding of x is denoted as $\llcorner x \lrcorner$ or simply x. Encoding

Task function Given two countable sets A and B representing the domain, a task can be represented as a function $f : A \to B$. Task

When not stated, A and B are implicitly encoded into $\{0, 1\}^*$.

Characteristic function Boolean function of form $f : \{0, 1\}^* \to \{0, 1\}$. Characteristic function

Given a characteristic function f, the language $\mathcal{L}_f = \{x \in \{0, 1\}^* \mid f(x) = 1\}$ can be defined.

Decision problem Given a language $\mathcal{M}$, a decision problem is the task of computing a boolean function f able to determine if a string belongs to $\mathcal{M}$ (i.e. $\mathcal{L}_f = \mathcal{M}$). Decision problem

1.1.3 Asymptotic notation

Big O A function $f : \mathbb{N} \to \mathbb{N}$ is $O(g)$ if g is an upper bound of f. Big O

$$f \in O(g) \iff \exists \bar{n} \in \mathbb{N} \text{ such that } \forall n > \bar{n}, \exists c \in \mathbb{R}^+ : f(n) \leq c \cdot g(n)$$

Big Omega A function $f : \mathbb{N} \to \mathbb{N}$ is $\Omega(g)$ if g is a lower bound of f. Big Omega

$$f \in \Omega(g) \iff \exists \bar{n} \in \mathbb{N} \text{ such that } \forall n > \bar{n}, \exists c \in \mathbb{R}^+ : f(n) \geq c \cdot g(n)$$

Big Theta A function $f : \mathbb{N} \to \mathbb{N}$ is $\Theta(g)$ if g is both an upper and lower bound of f. Big Theta

$$f \in \Theta(g) \iff f \in O(g) \text{ and } f \in \Omega(g)$$

2 Turing Machine

2.1 k-tape Turing Machine

Tape Infinite one-directional line of cells. Each cell can hold a symbol from a finite alphabet Γ. Tape

Tape head A tape head reads or writes one symbol at a time and can move left or right on the tape.

Input tape Read-only tape where the input will be loaded.

Work tape Read-write auxiliary tape used during computation.

Output tape Read-write tape that will contain the output of the computation.

Remark. Sometimes the output tape is not necessary and the final state of the computation can be used to determine a boolean outcome.

Instructions Given a finite set of states Q, at each step, a machine can: Instructions

Read from the k tape heads.

Replace the symbols under the writable tape heads, or leave them unchanged.

Change state.

Move each of the k tape heads to the left or right, or leave unchanged.

k-tape Turing Machine (TM) A Turing Machine working on k tapes (one of which is the input tape) is a triple (Γ, Q, δ): k-tape Turing Machine (TM)

- Γ is a finite set of tape symbols. We assume that it contains a blank symbol ($\square$), a start symbol ($\triangleright$), and the digits 0, 1.
- Q is a finite set of states. The initial state is q_{init} and the final state is q_{halt}.
- δ is the transition function that describes the instructions allowed at each step. It is defined as:
$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$$

  By convention, when the state is q_{halt}, the machine is stuck (i.e. it cannot change state or operate on the tapes):
$$\delta(q_{\text{halt}}, \{\sigma_1, \dots, \sigma_k\}) = \big(q_{\text{halt}}, \{\sigma_1, \dots, \sigma_k\}, (\mathtt{S}, \dots, \mathtt{S})\big)$$

Theorem 2.1.1 (Turing Machine equivalence)**.** The following computational models have, with at most a polynomial overhead, the same expressive power: 1-tape TMs, k-tape TMs, non-deterministic TMs, random access machines, λ-calculus, unlimited register machines, programming languages (Böhm-Jacopini theorem), ...

2.2 Computation

Configuration Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$, a configuration C is described by:

Configuration

- The current state q.
- The content of the tapes.
- The position of the tape heads.

Initial configuration Given the input $x \in \{0,1\}^*$, the initial configuration $\mathcal{I}_x$ is described as follows:

- The current state is q_{init}.
- The first (input) tape contains $\triangleright x\square\dots$. The other tapes contain $\triangleright\square\dots$.
- The tape heads are positioned on the first symbol of each tape.

Final configuration Given an output $y \in \{0,1\}^*$, the final configuration is described as follows:

- The current state is q_{halt}.
- The output tape contains $\triangleright y\square\dots$.

Computation (string) Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$, $\mathcal{M}$ returns $y \in \{0,1\}^*$ on input $x \in \{0,1\}^*$ (i.e. $\mathcal{M}(x) = y$) in t steps if:

Computation (string)

$$\mathcal{I}_x \xrightarrow{\delta} C_1 \xrightarrow{\delta} \dots \xrightarrow{\delta} C_t$$

where C_t is a final configuration for y.

Computation (function) Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$ and a function $f : \{0,1\}^* \to \{0,1\}^*$, $\mathcal{M}$ computes f iff:

Computation (function)

$$\forall x \in \{0,1\}^* : \mathcal{M}(x) = f(x)$$

If this holds, f is a computable function.

Computation in time T Given a TM $\mathcal{M}$ and the functions $f : \{0,1\}^* \to \{0,1\}^*$ and $T : \mathbb{N} \to \mathbb{N}$, $\mathcal{M}$ computes f in time T iff:

Computation in time T

$$\forall x \in \{0,1\}^* : \mathcal{M}(x) \text{ returns } f(x) \text{ in at most } T(|x|) \text{ steps}$$

Decidability in time T Given a function $f : \{0,1\}^* \to \{0,1\}$, the language $\mathcal{L}_f$ is decidable in time T iff f is computable in time T.

Decidability in time T

2.3 Universal Turing Machine

Turing Machine encoding Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$, the entire machine can be described by δ through tuples of form:

$$Q \times \Gamma^k \times Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$$

It is therefore possible to encode δ into a binary string and consequently create an encoding $\llcorner\mathcal{M}\lrcorner$ of $\mathcal{M}$.

The encoding should satisfy the following conditions:

1. For every $x \in \{0,1\}^*$, there exists a TM $\mathcal{M}$ such that $x = \llcorner\mathcal{M}\lrcorner$.

2. Every TM is represented by an infinite number of strings. One of them is the canonical representation.

Theorem 2.3.1 (Universal Turing Machine (UTM))**.** There exists a TM $\mathcal{U}$ such that, for every binary strings x and α, it emulates the TM defined by α on input x:

Universal Turing Machine (UTM)

$$\mathcal{U}(x, \alpha) = \mathcal{M}_\alpha(x)$$

where $\mathcal{M}_\alpha$ is the TM defined by α.

Moreover, $\mathcal{U}$ simulates $\mathcal{M}_\alpha$ with at most $CT\log(T)$ time overhead, where C only depends on $\mathcal{M}_\alpha$.

2.4 Computability

2.4.1 Undecidable functions

Theorem 2.4.1 (Existance of uncomputable functions)**.** There exists a function $uc : \{0,1\}^* \to \{0,1\}^*$ that is not computable by any TM.

Uncomputable functions

Proof. Consider the following function:

$$uc(\alpha) = \begin{cases} 0 & \text{if } \mathcal{M}_\alpha(\alpha) = 1 \\ 1 & \text{if } \mathcal{M}_\alpha(\alpha) \neq 1 \end{cases}$$

If uc was computable, there would be a TM $\mathcal{M}$ that computes it (i.e. $\forall \alpha \in \{0,1\}^* : \mathcal{M}(\alpha) = uc(\alpha)$). This will result in a contradiction:

$$uc(\llcorner \mathcal{M} \lrcorner) = 0 \iff \mathcal{M}(\llcorner \mathcal{M} \lrcorner) = 1 \iff uc(\llcorner \mathcal{M} \lrcorner) = 1$$

Therefore, uc cannot be computed. □

Halting problem Given an encoded TM α and a string x, the halting problem aims to determine if $\mathcal{M}_\alpha$ terminates on input x. In other words:

Halting problem

$$\texttt{halt}(\llcorner(\alpha, x)\lrcorner) = \begin{cases} 1 & \text{if } \mathcal{M}_\alpha \text{ stops on input } x \\ 0 & \text{otherwise} \end{cases}$$

Theorem 2.4.2. The halting problem is undecidable.

Proof. Note: this proof is slightly different from the traditional proof of the halting problem.

Assume that `halt` is decidable. Therefore, there exists a TM $\mathcal{M}_{\texttt{halt}}$ that decides it.

We can define a new TM $\mathcal{M}_{uc}$ that uses $\mathcal{M}_{\texttt{halt}}$ such that:

$$\mathcal{M}_{uc}(\alpha) = \begin{cases} 1 & \text{if } \mathcal{M}_{\texttt{halt}}(\alpha, \alpha) = 0 \text{ (i.e. } \mathcal{M}_\alpha(\alpha) \text{ diverges)} \\ \begin{cases} 0 & \text{if } \mathcal{M}_\alpha(\alpha) = 1 \\ 1 & \text{if } \mathcal{M}_\alpha(\alpha) \neq 1 \end{cases} & \text{if } \mathcal{M}_{\texttt{halt}}(\alpha, \alpha) = 1 \text{ (i.e. } \mathcal{M}_\alpha(\alpha) \text{ converges)} \end{cases}$$

This results in a contradiction:

- $\mathcal{M}_{uc}(\llcorner \mathcal{M}_{uc} \lrcorner) = 1 \Leftarrow \mathcal{M}_{\texttt{halt}}(\llcorner \mathcal{M}_{uc} \lrcorner, \llcorner \mathcal{M}_{uc} \lrcorner) = 0 \iff \mathcal{M}_{uc}(\llcorner \mathcal{M}_{uc} \lrcorner)$ diverges
- $\mathcal{M}_{\texttt{halt}}(\llcorner \mathcal{M}_{uc} \lrcorner, \llcorner \mathcal{M}_{uc} \lrcorner) = 1 \Rightarrow \mathcal{M}_{uc}$ is not computable by Theorem 2.4.1.

□

Diophantine equation Polynomial equality with integer coefficients and a finite number of unknowns.

Diophantine equation

Theorem 2.4.3 (MDPR)**.** Determining if an arbitrary diophantine equation has a solution is undecidable.

2.4.2 Rice's theorem

Semantic language Given a language $\mathcal{L} \subseteq \{0,1\}^*$, $\mathcal{L}$ is semantic if:

Semantic language

- Any string in $\mathcal{L}$ is an encoding of a TM.
- If $\llcorner \mathcal{M} \lrcorner \in \mathcal{L}$ and the TM $\mathcal{N}$ computes the same function of $\mathcal{M}$, then $\llcorner \mathcal{N} \lrcorner \in \mathcal{L}$.

A semantic language can be seen as a set of TMs that have the same property.

Trivial language A language $\mathcal{L}$ is trivial iff $\mathcal{L} = \varnothing$ or $\mathcal{L} = \{0,1\}^*$

Theorem 2.4.4 (Rice's theorem)**.** If a semantic language is non-trivial, then it is undecidable (i.e. any decidable semantic language is trivial).

Rice's theorem

Proof idea. Assuming that there exists a non-trivial decidable semantic language $\mathcal{L}$, it is possible to prove that the halting problem is decidable. Therefore, $\mathcal{L}$ is undecidable. □

3 Complexity

Complexity class Set of tasks that can be computed within some fixed resource bounds. Complexity class

3.1 Polynomial time

Deterministic time (DTIME) Let $T : \mathbb{N} \to \mathbb{N}$ and $\mathcal{L}$ be a language. $\mathcal{L}$ is in $\mathbf{DTIME}(T(n))$ iff there exists a TM that decides $\mathcal{L}$ in time $O(T(n))$. Deterministic time ($\mathbf{DTIME}$)

Polynomial time (P) The class $\mathbf{P}$ contains all the tasks computable in polynomial time: Polynomial time ($\mathbf{P}$)

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c)$$

Remark. $\mathbf{P}$ is closed to various operations on programs (e.g. composition of programs)

Remark. In practice, the exponent is often small.

Remark. $\mathbf{P}$ considers the worst case and is not always realistic. Other alternative computational models exist.

Church-Turing thesis Any physically realizable computer can be simulated by a TM with an arbitrary time overhead. Church-Turing thesis

Strong Church-Turing thesis Any physically realizable computer can be simulated by a TM with a polynomial time overhead. Strong Church-Turing thesis

Remark. If this thesis holds, the class $\mathbf{P}$ is robust (i.e. does not depend on the computational device) and is therefore the smallest class of bounds.

Deterministic time for functions (FDTIME) Let $T : \mathbb{N} \to \mathbb{N}$ and $f : \{0,1\}^* \to \{0,1\}^*$. f is in $\mathbf{FDTIME}(T(n))$ iff there exists a TM that computes it in time $O(T(n))$. Deterministic time for functions ($\mathbf{FDTIME}$)

Polynomial time for functions (FP) The class $\mathbf{FP}$ is defined as: Polynomial time for functions ($\mathbf{FP}$)

$$\mathbf{FP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(n^c)$$

Remark. It holds that $\forall \mathcal{L} \in \mathbf{P} \Rightarrow f_{\mathcal{L}} \in \mathbf{FP}$, where $f_{\mathcal{L}}$ is the characteristic function of $\mathcal{L}$. Generally, the contrary does not hold.

3.2 Exponential time

Exponential time (EXP/FEXP) The $\mathbf{EXP}$ and $\mathbf{FEXP}$ classes are defined as: Exponential time ($\mathbf{EXP}/\mathbf{FEXP}$)

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c}) \qquad \mathbf{FEXP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(2^{n^c})$$

Theorem 3.2.1. The following hold:

$$\mathbf{P} \subset \mathbf{EXP} \qquad \mathbf{FP} \subset \mathbf{FEXP}$$

3.3 NP class

Certificate Given a set of pairs $\mathcal{C}_\mathcal{L}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$, we can define the language $\mathcal{L}$ such that: Certificate

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)} : (x,y) \in \mathcal{C}_\mathcal{L}\}$$

Given a string w and a certificate y, we can exploit $\mathcal{C}_\mathcal{L}$ as a test to check whether y is a certificate for w:

$$w \in \mathcal{L} \iff (w,y) \in \mathcal{C}_\mathcal{L}$$

Nondeterministic TM (NDTM) TM that has two transition functions δ_0, δ_1 and, at each step, non-deterministically chooses which one to follow. A state q_{accept} is always present: Nondeterministic TM (NDTM)

- A NDTM accepts a string iff one of the possible computations reaches q_{accept}.
- A NDTM rejects a string iff none of the possible computations reach q_{accept}.

Nondeterministic time (NDTIME) Let $T : \mathbb{N} \to \mathbb{N}$ and $\mathcal{L}$ be a language. $\mathcal{L}$ is in $\mathbf{NDTIME}(T(n))$ iff there exists a NDTM that decides $\mathcal{L}$ in time $O(T(n))$. Nondeterministic time (**NDTIME**)

Remark. A NDTM $\mathcal{M}$ runs in time $T : \mathbb{N} \to \mathbb{N}$ iff for every input, any possible computation terminates in time $O(T(n))$.

Complexity class NP Complexity class **NP**

NDTM formulation The class **NP** contains all the tasks computable in polynomial time by a nondeterministic TM:

$$\mathbf{NP} = \bigcup_{c \geq 1} \mathbf{NDTIME}(n^c)$$

Verifier formulation Let $\mathcal{L} \in \{0,1\}^*$ be a language. $\mathcal{L}$ is in **NP** iff there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial TM $\mathcal{M}$ (verifier) such that:

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)} : \mathcal{M}(\llcorner(x,y)\lrcorner) = 1\}$$

In other words, $\mathcal{L}$ is the language of the strings that can be verified by $\mathcal{M}$ in polynomial time using a certificate y of polynomial length.

Theorem 3.3.1. $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$.

Proof. We have to prove that $\mathbf{P} \subseteq \mathbf{NP}$ and $\mathbf{NP} \subseteq \mathbf{EXP}$:

$\mathbf{P} \subseteq \mathbf{NP}$) Given a language $\mathcal{L} \in \mathbf{P}$, we want to prove that $\mathcal{L} \in \mathbf{NP}$.

By hypothesis, there is a polynomial time TM $\mathcal{N}$ that decides $\mathcal{L}$. To prove that $\mathcal{L}$ is in **NP**, we show that there is a polynomial verifier $\mathcal{M}$ that certifies $\mathcal{L}$ with a polynomial certificate. We can use any constant certificate (e.g. of length 1) and use $\mathcal{N}$ as the verifier $\mathcal{M}$:

$$\mathcal{M}(x,y) = \begin{cases} 1 & \text{if } \mathcal{N}(x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{M}$ can ignore the polynomial certificate and it "verifies" a string in polynomial time through $\mathcal{N}$.

NP $\subseteq$ EXP) Given a language $\mathcal{L} \in \mathbf{NP}$, we want to prove that $\mathcal{L} \in \mathbf{EXP}$.

By hypothesis, there is a polynomial time TM $\mathcal{N}$ that is able to certify any string in $\mathcal{L}$ with a polynomial certificate. Given a polynomial p, can define the following algorithm:

```
def np_to_exp(x ∈ {0,1}*):
    foreach y ∈ {0,1}^p(|x|):
        if M(x,y) == 1:
            return 1
    return 0
```

The algorithm has complexity $O(2^{p(|x|)}) \cdot O(q(|x| + |y|)) = O(2^{p(|x|)+\log(q(|x|+|y|))})$, where q is a polynomial. Therefore, the complexity is exponential.

$\square$

Polynomial-time reducibility A language $\mathcal{L}$ is poly-time reducible to $\mathcal{H}$ ($\mathcal{L} \leq_p \mathcal{H}$) iff:

Polynomial-time reducibility

$$\exists f : \{0,1\}^* \to \{0,1\}^* \text{ such that } (x \in \mathcal{L} \iff f(x) \in \mathcal{H}) \text{ and } \\ f \text{ is computable in poly-time}$$

f can be seen as a mapping function.

Remark. Intuitively, when $\mathcal{L} \leq_p \mathcal{H}$, $\mathcal{H}$ is at least as difficult as $\mathcal{L}$.

Theorem 3.3.2. The relation $\leq_p$ is a pre-order (i.e. reflexive and transitive).

Proof. We want to prove that $\leq_p$ is reflexive and transitive:

Reflexive) Given a language $\mathcal{L}$, we want to prove that $\mathcal{L} \leq_p \mathcal{L}$.

We have to find a poly-time function $f : \{0,1\}^* \to \{0,1\}^*$ such that:

$$x \in \mathcal{L} \iff f(x) \in \mathcal{L}$$

We can choose f as the identity function.

Transitive) Given the languages $\mathcal{L}, \mathcal{H}, \mathcal{J}$, we want to prove that:

$$(\mathcal{L} \leq_p \mathcal{H}) \land (\mathcal{H} \leq_p \mathcal{J}) \Rightarrow (\mathcal{L} \leq_p \mathcal{J})$$

By hypothesis, it holds that $\mathcal{L} \leq_p \mathcal{H}$ and $\mathcal{H} \leq_p \mathcal{J}$. Therefore, there are two poly-time functions $f, g : \{0,1\}^* \to \{0,1\}^*$ such that:

$$x \in \mathcal{L} \iff f(x) \in \mathcal{H} \text{ and } y \in \mathcal{H} \iff f(y) \in \mathcal{J}$$

We want to find a poly-time mapping from $\mathcal{L}$ to $\mathcal{J}$. This function can be the composition $(g \circ f)(z) = g(f(z))$. $(g \circ f)$ is poly-time as f and g are poly-time.

$\square$

NP-hard Given a language $\mathcal{H} \in \{0,1\}^*$, $\mathcal{H}$ is **NP**-hard iff:

NP-hard

$$\forall \mathcal{L} \in \mathbf{NP} : \mathcal{L} \leq_p \mathcal{H}$$

NP-complete Given a language $\mathcal{H} \in \{0,1\}^*$, $\mathcal{H}$ is **NP**-complete iff:

NP-complete

$$\mathcal{H} \in \mathbf{NP} \text{ and } \mathcal{H} \text{ is } \mathbf{NP}\text{-hard}$$

Theorem 3.3.3.

1. If $\mathcal{L}$ is **NP**-hard and $\mathcal{L} \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.

2. If $\mathcal{L}$ is **NP**-complete, then $\mathcal{L} \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$.

Proof.

1. Let $\mathcal{L}$ be **NP**-hard and $\mathcal{L} \in \mathbf{P}$. We want to prove that $\mathbf{P} = \mathbf{NP}$:

   $\mathsf{P} \subseteq \mathsf{NP}$) Proved in Theorem 3.3.1.

   $\mathsf{NP} \subseteq \mathsf{P}$) Let $\mathcal{H}$ be a language in **NP**. As $\mathcal{L}$ is **NP**-hard, by definition it holds that $\mathcal{H} \leq_p \mathcal{L}$. Moreover, by hypothesis, it holds that $\mathcal{L} \in \mathbf{P}$. Therefore, we can conclude that $\mathcal{H} \in \mathbf{P}$ as it can be reduced to a language in **P**.

2. Let $\mathcal{L}$ be **NP**-complete. We want to prove that $\mathcal{L} \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$:

   $(\mathcal{L} \in \mathsf{P}) \Rightarrow (\mathsf{P} = \mathsf{NP})$) Trivial for Point 1 as $\mathcal{L}$ is also **NP**-hard.

   $(\mathcal{L} \in \mathsf{P}) \Leftarrow (\mathsf{P} = \mathsf{NP})$) Let $\mathbf{P} = \mathbf{NP}$. As $\mathcal{L}$ is **NP**-complete, it holds that $\mathcal{L} \in \mathbf{NP} = \mathbf{P}$.

□

Theorem 3.3.4. The problem `TMSAT` of simulating any TM is **NP**-complete:

$$\texttt{TMSAT} = \{(\alpha, x, 1^n, 1^t) \mid \exists u \in \{0,1\}^n : \mathcal{M}_\alpha(x, u) = 1 \text{ within } t \text{ steps}\}$$

Theorem 3.3.5 (Cook-Levin)**.** The following languages are **NP**-complete:

Cook-Levin theorem

$$\begin{aligned} \texttt{SAT} &= \{\llcorner F \lrcorner \mid F \text{ is a satisfiable CNF}\} \\ \texttt{3SAT} &= \{\llcorner F \lrcorner \mid F \text{ is a satisfiable 3CNF}\} \end{aligned}$$