COS210 - Theoretical Computer Science
Decidable and Undecidable Languages (Part 1)

Decidability

Given our construction of a Turing machine, we want to consider the limitations of such a model of computation:

What is computable? What is **not** computable?

Computability is synonymous with the question of **decidability** in the context of Turing machines:

Can we construct a Turing machine M that can **decide** whether a string w is contained in a language A (**accept**), or is not contained in A (**reject**)?

We will see that for certain kinds of languages such a Turing machine can be constructed (**decidable languages**) whereas for other languages such a construction is not possible (**undecidable languages**)

Decidability

The class of decidable languages is defined as follows:

Definition (Decidable Language)

Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language.
A is **decidable**, if there exists a Turing machine M, such that for every string $w \in \Sigma^*$, the following holds:

- If $w \in A$, then the computation of the Turing machine M, on the input string w, **terminates** in the **accept** state.
- If $w \notin A$, then the computation of the Turing machine M, on the input string w, **terminates** in the **reject** state.

In particular, the constructed Turing machine M always terminates

Decidability – First Example

- Consider the simple language $L = \{11\}$ over $\Sigma = \{0, 1\}$
- We can easily build a Turing machine M with $L(M) = L$
- That is, M accepts 11 and rejects every other input over Σ
- Therefore, the language L is **decidable**

(Exercise: Build the Turing machine that accepts L)

Decidability in Context of Deterministic Finite-Automata

Are **all** languages L that can be described by some deterministic finite automaton M **decidable**?

This is the same as asking if the following **one** language is decidable:

$$A_{DFA} = \{\langle M, w\rangle : M \text{ is a } DFA \text{ and } w \in L(M)\}$$

- The language A_{DFA} consists of pairs $\langle M, w\rangle$ where M is a DFA and w is a string accepted by M
- A Turing machine that decides A_{DFA} must take pairs of the form $\langle M, w\rangle$ as an input
- We can think of an encoding of the DFA M such that the pair $\langle M, w\rangle$ can be written onto the tape of the Turing machine

Decidability in Context of General Computational Models

Are **all** languages L that can be described by some computational model C of type T **decidable**?

(The type can be for instance: DFA, NFA, CFG, PDA or TM)

Equivalently, is the following language decidable?

$$A_T = \{\langle C, w\rangle : C \text{ is of type } T \text{ and } w \in L(C)\}$$

We need to prove whether or not we can construct an algorithm (= Turing machine) that takes an encoded pair $u = \langle C, v\rangle$ as input and

- **terminates** in the state **accept** if $u \in A_T$
- **terminates** in the state **reject** if $u \notin A_T$

Decidability in Context of Deterministic Finite-Automata

Theorem (1)

The language A_{DFA} is decidable

We will prove this by defining an algorithm that accepts all inputs $u \in A_{DFA}$ and rejects all inputs $u \notin A_{DFA}$

Decidability in Context of Deterministic Finite-Automata

Theorem (1)

The language A_{DFA} is decidable

Proof:

On input u, check whether u is a **correct encoding** of a pair $\langle M, w\rangle$ where M is a DFA and w is a string

- If not, then terminate and **reject** u
- If yes, then **simulate** the run of the DFA M over w
  - If M accepts w, then terminate and **accept** u
  - If M does not accept w, then terminate and **reject** u

$\square$

Non-Deterministic Finite-Automata and Decidability

Theorem (2)

The language A_{NFA} is decidable, where

$$A_{NFA} = \{\langle M, w\rangle : M \text{ is an NFA and } w \in L(M)\}$$

Proof:

On input u, check whether u is a **correct encoding** of a pair $\langle M, w\rangle$ where M is an NFA and w is a string

- If not, then terminate and **reject** u
- If yes, then **convert** the NFA M to a DFA N
  <u>a run of an NFA is not unique, but a run of a DFA is</u>
  <u>we have seen that each NFA can be converted to an equivalent DFA</u>
  - If N accepts w, then terminate and **accept** u
  - If N does not accept w, then terminate and **reject** u

$\square$

Context-Free Grammars and Decidability

Theorem (3)

The language A_{CFG} is decidable, where

$$A_{CFG} = \{\langle G, w\rangle : G \text{ is a context-free grammar and } w \in L(G)\}$$

Proof:
On input u, check whether u is a **correct encoding** of a pair $\langle G, w\rangle$ where G is CFG and w is a string. If not, then reject u

If yes, then we need to decide whether $w \in L(G)$ or $w \notin L(G)$

- $w \in L(G)$ is equivalent to the derivation $S \overset{*}{\Longrightarrow} w$
- Our algorithm will first produce all 1-step derivations and check whether w could be derived. If not, it continues with all 2-step derivations etc.
- If $w \in L(G)$, then this algorithm with eventually terminate an can **accept** u

Context-Free Grammars and Decidability

Proof cont:

If $w \notin L(G)$, then we need to figure out after how many unsuccessful derivation steps we can stop an conclude that w is not derivable at all

Solution: We convert G to grammar G' in Chomsky normal form

- Let n be the length of w. If $w \in L(G')$, then $S' \overset{2n-1}{\Longrightarrow} w$

  The derivation of w from G' requires $2n-1$ steps:

  - From the single start variable to n variables takes $n-1$ steps
  - From the n variables to n terminals takes n additional steps

- If our algorithm cannot derive w from G' within $2n-1$ steps, then $w \notin L(G)$ and the algorithm can terminate and **reject** u

$\square$

Decidability of Context-Free Languages

Theorem (3)

The language A_{CFG} is decidable

This implies:

Theorem (4)

Every **context-free language** *is decidable*

Proof:

- Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be an arbitrary **context-free language**
- There exists a **context-free grammar** G' in Chomsky normal form with $L(G') = A$
- Given an arbitrary string $w \in \Sigma^*$, we have seen how to **algorithmically decide** whether w can be derived from G' or not $\square$

Decidability of Regular Languages

Theorem (4)

Every **context-free language** *is decidable*

This implies:

Theorem (5)

Every **regular language** *is decidable*

Proof:

The set of **regular languages** is a **subset** of the set of **context free languages** □

Turing Machines and Decidability?

Are all languages that can be accepted by Turing machines also decidable?

- This reduces to the question whether the following language is decidable or not

$$A_{TM} = \{\langle M, w\rangle : M \text{ is a Turing machine and } w \in L(M)\}$$

Turing Machines and Decidability?

Are all languages that can be accepted by Turing machines also decidable?

- This reduces to the question whether the following language is decidable or not

$$A_{TM} = \{\langle M, w\rangle : M \text{ is a Turing machine and } w \in L(M)\}$$

- In fact, the language A_{TM} is **undecidable**
- There is no algorithm that, when given an arbitrary Turing machine M and an arbitrary input string w for M, decides in a finite amount of time, whether or not M accepts w

Turing Machines and Undecidability

Theorem (6)

The language A_{TM} is **undecidable**, *where*

$$A_{TM} = \{\langle M, w\rangle : M \textit{ is a Turing machine and } w \in L(M)\}$$

We will proof this by showing that an algorithm that decides A_{TM} in a finite amount of time cannot exist

Proof by Contradiction:

Assume that A_{TM} is decidable

Then there exists a Turing machine H such that for every input $\langle M, w \rangle$:

- If $\langle M, w \rangle \in A_{TM}$ (M accepts w),
  then H terminates in **accept** state
- if $\langle M, w \rangle \notin A_{TM}$ (M rejects w or does not terminate on w),
  then H terminates in **reject** state

In particular, H **always terminates**

Turing Machines and Undecidability

Proof cont:

We have that H always terminates

We now define another Turing machine D:

- D takes a Turing machine $\langle M \rangle$ as an input
- Then D simulates the machine H on input $\langle M, \langle M \rangle \rangle$
  (This will answer whether M accepts itself as an input or not)
  - If H **accepts** $\langle M, \langle M \rangle \rangle$ (M accepts itself)
    then D terminates in **reject** state
  - If H **rejects** $\langle M, \langle M \rangle \rangle$ (M rejects itself or never terminates on input M)
    then D terminates in **accept** state

Also D **always terminates**

Proof cont:

For any input $\langle M \rangle$ of the Turing machine D:

- If M **accepts** $\langle M \rangle$, then D **rejects** $\langle M \rangle$
- If M **rejects** or **not terminates** on $\langle M \rangle$, then D **accepts** $\langle M \rangle$

Turing Machines and Undecidability

Proof cont:

For any input $\langle M \rangle$ of the Turing machine D:

- If M **accepts** $\langle M \rangle$, then D **rejects** $\langle M \rangle$
- If M **rejects** or **not terminates** on $\langle M \rangle$, then D **accepts** $\langle M \rangle$

Now what happens if we use $\langle D \rangle$ as the input of D?

Proof cont:

For any input $\langle M \rangle$ of the Turing machine D:

- If M **accepts** $\langle M \rangle$, then D **rejects** $\langle M \rangle$
- If M **rejects** or **not terminates** on $\langle M \rangle$, then D **accepts** $\langle M \rangle$

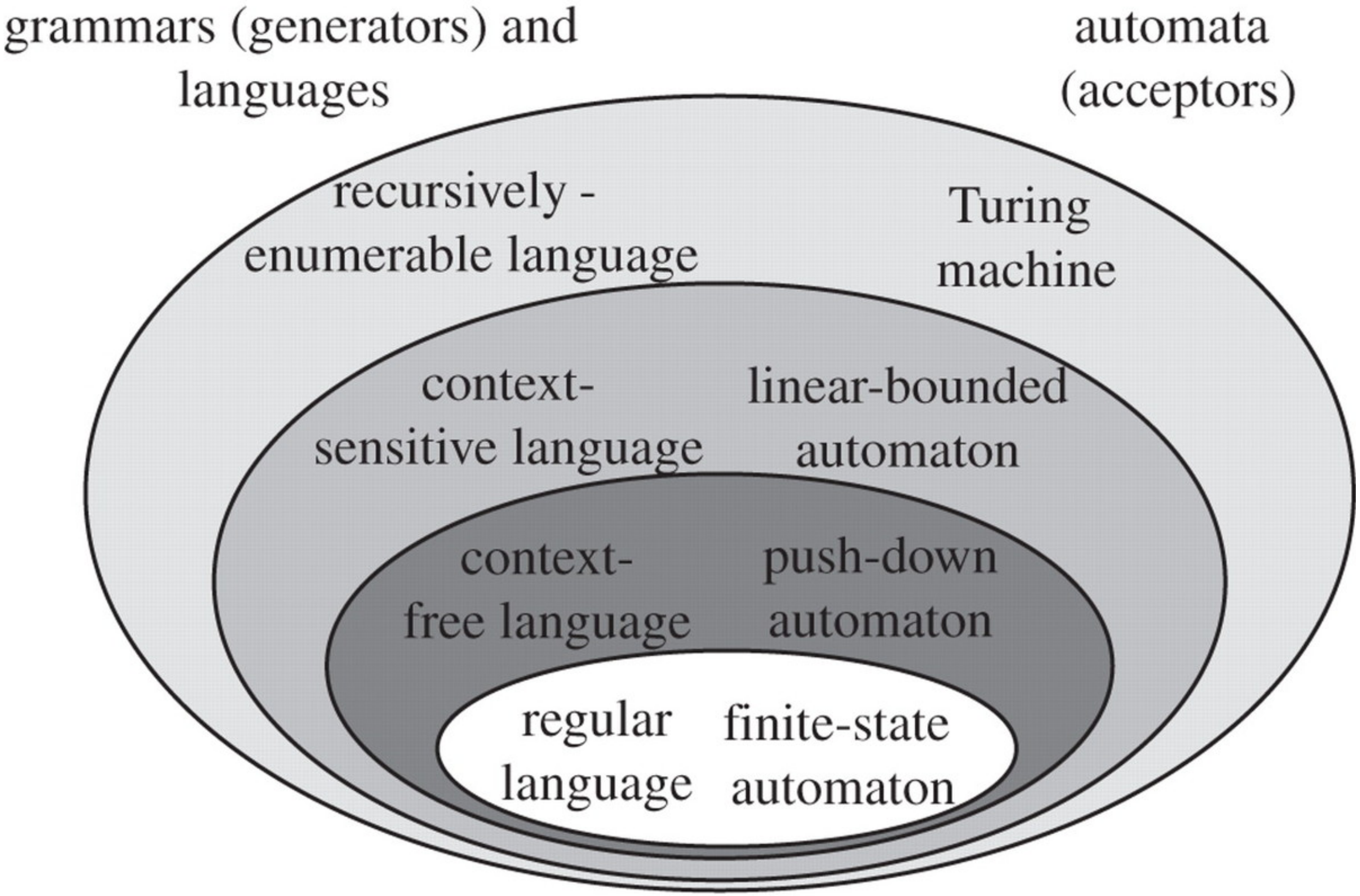
Now what happens if we use $\langle D \rangle$ as the input of D?

- If D **accepts** $\langle D \rangle$, then D **rejects** $\langle D \rangle$
- If D **rejects** ~~or never terminates on~~ $\langle D \rangle$, then D **accepts** $\langle D \rangle$

Contradiction. Therefore, the machine H that decides A_{TM} cannot exist

Consequently, the language A_{TM} is **undecidable** $\square$

Turing Machines and Undecidability



the traditional Chomsky hierarchy