Decidability

Last Updated: August 18th, 2026

1 Introduction

Perhaps the most fundamental question one can ask about the Turing machine model of computation is "which computational problems are solvable using a Turing machine?". In what follows we restrict attention to decision problems, since by now it should be understood that associated with most non-decision problems are decision problems that capture the essence of their non-decision counterparts. The question then becomes "which computational problems are Turing decidable?". In other words, is there a Turing machine that halts on all inputs and accepts only those inputs that represent positive instances of the decision problem in question? If yes, then we say that the decison problem is **decidable**. Otherwise we say that it is **undecidable**.

Moving forward, we simply refer to a computational decsion problem as a "decision problem". Moreover, we know from the Church-Turing thesis, that establishing the decidability of a decision problem L amounts to being able to describe a deterministic step-by-step procedure that, when applied to instance x of L, results in a final decision as to whether or not x is a positive instance of L. Moreover, the following are three different levels of abstraction that are often used to describe such a procedure.

Natural Language The procedure is written in a somewhat informal manner that uses one or more sentences of a natural language along with any necessary mathematical formalism.

Pseudocode The procedure is written in a semiformal manner as a structured list of statements, where each statement may be expressed using a combination of natural language, program-like syntax, and mathematical symbolism.

Formal Program The procdure is described using a programming language that usually consists of structured lines of syntax.

In theoretical computer science, formal programs are rarely used except for the purpose of demonstration.

The vast majority of computational decision problems that are encountered in both applied and theoretical computer science are decidable. Indeed, computing professionals care mostly about computational decision problems that have a programming solution, and therefore are decidable.

Quite often it can be seen that a decision problem can always be (oftentimes naively) solved by performing a finite number of steps. Moreover, if there is a way to organize these steps into a step-by-step process, then the problem is guaranteed to be decidable.

Example 1.1. For example, deciding if a natural number $n \geq 2$ is prime is decidable, since the number of steps needed to check is finite: divide each of $2, \ldots, n-1$ into n and make sure that none of them divide evenly into n. For large n, this is a very inefficient algorithm, but when addressing the question of decidability, algorithm efficiency doesn't matter so long as the algorithm is correct and guaranteed to terminate. □

We may generalize the previous example and state that
"A decision problem is decidable if, for each problem instance, there is always a finite number of possibilities that must be checked and the process of checking a single possibility always requires at most a finite number of steps".

Thus, from the vantage point of decidability, the only interesting problems are those for which there may be an infinite number of possible solutions and/or for which checking a possible solution may require an unlimited number of steps.

Example 1.2. A **Diophantine equation** is one of the form $p(x_1, \ldots, x_m) = 0$, where $p(x_1, \ldots, x_m)$ is a polynomial in $m \geq .1$ variables and for which the term coefficients are integers. For example, if $m = 3$, $x_1 = x$, $x_2 = y$, and $x_3 = z$, then

$$xy^2 - x^2 - 8xy^2z + x^3yz^4 - 7xz = 0$$

is a Diophantine equation. An instance of the **Diophantine Equation** decision problem is a Diophantine Equation, and the problem is to decide if there is an integer assignment that can be made to the equation variables that results in the left side of the equation evaluating to 0. Notice that each of the variables of a Diophantine equation has an infinite domain equal to the set of all integers I. Thus, it is unclear if the problem can be decided in a finite number of steps, since there are infinitely many possible assignments.

The following result is considered one of the crowning achivemennts of theoretical computer science.

Theorem 1.3. (Yuri Matiyasevich, 1970) The Diophantine Equation decision problem is undecidable.

In this lecture we examine several decision problems that are related to the models of computation that have been studied in this course: DFA's, NFA's, regular expressions, CFL's, PDA's, and Turing Machines.

2 Decision Problems Related to DFA's

Definition 2.1. An instance of $\texttt{Accept}_{\texttt{DFA}}$ consists of a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and a word w over M's input alphabet Σ. The problem is to decide if M accepts w.

Theorem 2.2. $\texttt{Accept}_{\texttt{DFA}}$ is decidable.

Proof of Theorem. The following pseudocode describes a procedure for deciding $\texttt{Accept}_{\texttt{DFA}}$.

Input: a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and a word w over M's input alphabet Σ.

Output: 1 iff M accepts w.

$q \leftarrow q_0$ //Initialize current state to be the initial state.

For each $i \in \{1, \ldots, |w|\}$,

$q \leftarrow \delta(q, w_i)$. //Use δ-transition function to compute next state.

Return $q \in F$. //Accept iff q is an accepting state.

Corollary 2.3. The following problems are also decidable.

1. $\texttt{Accept}_{\texttt{NFA}}$
2. $\texttt{Accept}_{\texttt{RegEx}}$

For example, an instance of $\texttt{Accept}_{\texttt{RegEx}}$ consists of a regular expression E over some alphabet Σ and a word w over Σ. The problem is to decide if $w \in L(E)$.

Definition 2.4. An instance of $\texttt{Empty}_{\texttt{DFA}}$ consists of a DFA $M = (Q, \Sigma, \delta, q_0, F)$. The problem is to decide if $L(M) = \emptyset$.

Theorem 2.5. $\texttt{Empty}_{\texttt{DFA}}$ is decidable.

Proof of Theorem. The following pseudocode describes a procedure for deciding $\texttt{Empty}_{\texttt{DFA}}$.

Input: a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

Output: 1 iff M accepts w.

Construct the directed graph $G = (Q, E)$, where

$(q_1, q_2) \in E$ iff there is some symbol s for which $\delta(q_1, s) = q_2$.

For each $q \in F$,

If $\texttt{Reachable}(G, q_0, q)$, then reject. //There is at least one word accepted by M.

Return 1.

Corollary 2.6. The following problems are also decidable.

1. $\texttt{Empty}_{\texttt{NFA}}$
2. $\texttt{Empty}_{\texttt{RegEx}}$

$A \oplus B$

Definition 2.7. An instance of $\texttt{Equal}_{\texttt{DFA}}$ consists of two DFAs $M_1 = (Q, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$. The problem is to decide if $L(M_1) = L(M_2)$.

Theorem 2.8. $\texttt{Equal}_{\texttt{DFA}}$ is decidable.

$L(M_1) \oplus L(M_2) = \emptyset$

Proof of Theorem. In the Finite Automata lecture it was established that regular languages are closed under the union, intersection, complement, symmetric difference, and difference operations. Thus, given two DFA's M_1 and M_2 we may construct a DFA M that accepts $L(M_1) \oplus L(M_2)$. Moreover $L(M_1) = L(M_2)$ iff $L(M) = \emptyset$, which is decidable by Theorem 2. □

Corollary 2.9. The following problems are also decidable.

1. $\texttt{Equal}_{\texttt{NFA}}$
2. $\texttt{Equal}_{\texttt{RegEx}}$

De Morgan's

$A \cap B = \overline{\overline{A} \cup \overline{B}}$

$L(M_1)$ $L(M_2)$

$\Rightarrow$ closure under intersection

$L(M_1) = L(M_2) \iff$

$\overline{L(M_1) \cap L(M_2) = \emptyset}$ and $\overline{L(M_2) \cap L(M_1) = \emptyset}$

Decision Problems Related to CFG's and PDA's

Definition 2.10. An instance of $\texttt{Accept}_{\texttt{CFG}}$ consists of a context free grammar $G = (V, \Sigma, R, S)$ and a word w over Σ. The problem is to decide if $w \in L(G)$, the language derived by G.

Theorem 2.11. $\texttt{Accept}_{\texttt{CFG}}$ is decidable.

Proof of Theorem. The following pseudocode describes a recursive algorithm for deciding $\texttt{Accept}_{\texttt{CFG}}$. The code makes use of the notation $u[\geq i]$ which equals the suffix of word u starting at the ith letter of u. In case $i > |u|$, then $u[\geq i]$ is assigned ε. Note: for the initial call to `accept`, ν is assigned S, the start variable of G, and we assume that w is nonempty (for $w = \varepsilon$ deciding if w is accepted by G amounts to checking if start variable S has an ε rule). Note that w may become empty in subsequent calls to `accept`.

To better understand the algorithm, it helps to recall that a CFG in Chomsky Normal form has the following properties: i) only the start varible S is allowed to have an ε rule, ii) the start variable does not appear in the body of any rule, iii) a rule body must either consist of two variables or one terminal.

accept(S_0, ab)

Name: `accept`

Input: a CFG $G = (V, \Sigma, R, S)$ in Chomsky Normal Form, a word w over Σ, and a sequence of variables ν.

Output: 1 iff ~~M accepts w.~~ ν derives w

If $\nu == \varepsilon$, then return $(w == \varepsilon)$.

Let variable $A = \nu_1$ be the first variable of ν.

For each rule $\rho \in R$ for which $\text{head}(\rho) = A$,

If $\text{body}(\rho) = s \in \Sigma$ and $s = w_1$ and $\text{accept}(G, w[\geq 2], \nu[\geq 2])$, then return 1.
Else if $\text{body}(\rho) = BC$, where $B, C \in V$, and $|\nu| < |w|$ and $\text{accept}(G, w, BC \circ \nu[\geq 2])$,
then return 1.

Return 0. //All possible rule combinations have been tried and were unsuccessful in deriving w.

The above algorithm is initially called with $\nu = S$, and always terminates since $|\nu|$ is never allowed to exceed the length of input word w. Moreover, since the rules of G are being followed, if w is accepted, then $w \in L(G)$. Conversely, if $w \in L(G)$, then there is at least one leftmost derivation that derives w and one can prove using induction that such a correct leftmost derivation will eventually be realized by the recursive algorithm, leading to the acceptance of w. □

Corollary 2.12. The problem $\texttt{Accept}_{\texttt{PDA}}$ is decidable, where an instance of $\texttt{Accept}_{\texttt{PDA}}$ consists of a PDA M with input alphabet Σ and a word w over Σ. The problem is to decide if $w \in L(M)$.

Example 2.13. The CFG with rule set

$$S \to SS \mid aSb \mid \varepsilon$$

converted to Chomsky normal form yields the new rule set

$$S_0 \to SS \mid AB \mid WB \mid \varepsilon$$

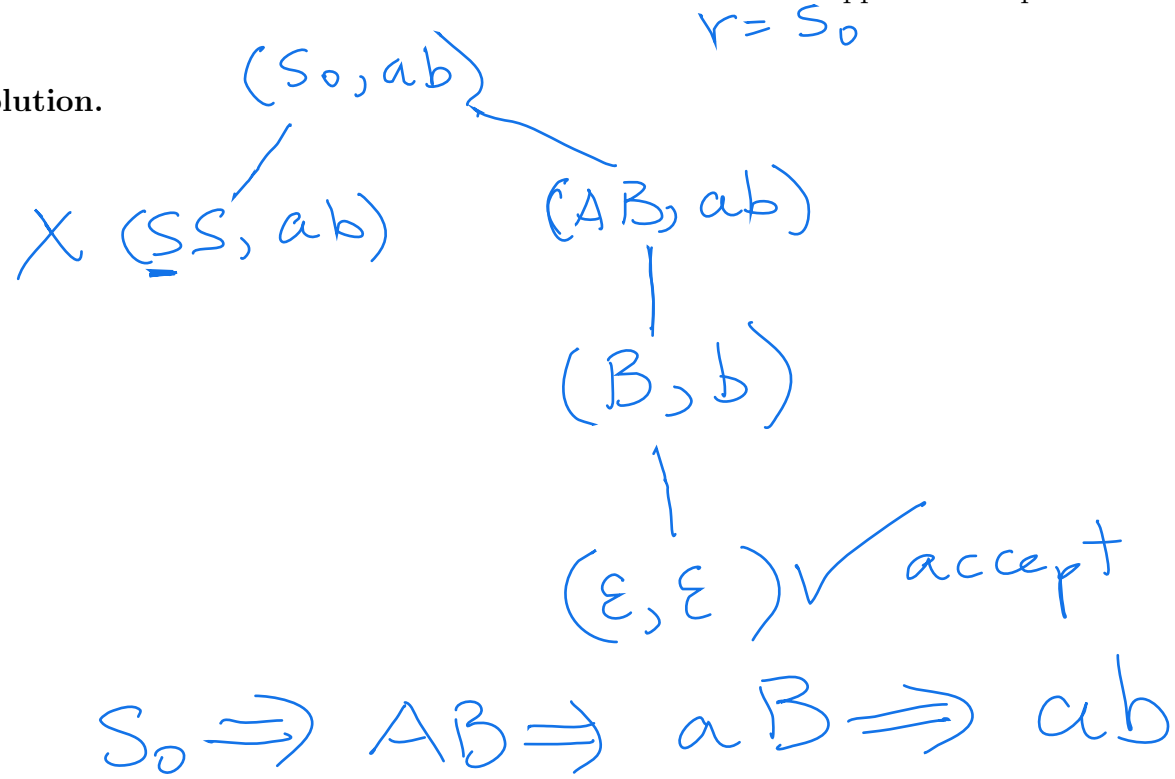
$$S \to SS \mid AB \mid WB$$

$$W \to AS$$

$$A \to a \quad B \to b$$

Demonstrate the `accept` algorithm above with this new rule set and with the input word $w = ab$. Draw the entire recursion tree and label each node with the pair (ν, w) where ν is the current variable word and w is the current input word at that point of the recursion. When replacing a variable with one of its bodies make sure to follow the order that the bodies appear in the provided list of rules.

Solution.



$S_0 \Rightarrow AB \Rightarrow aB \Rightarrow ab$

Definition 2.14. An instance of $\texttt{Empty}_{\texttt{CFG}}$ is a context free grammar $G = (V, \Sigma, R, S)$ and the problem is to decide if $L(G) = \emptyset$.

Theorem 2.15. $\texttt{Empty}_{\texttt{CFG}}$ is decidable.

Proof of Theorem.

Input: a CFG $G = (V, \Sigma, R, S)$.

Output: 1 iff $L(G) = \emptyset$.

Mark all terminal symbols of Σ.

Set marked="true".

While marked equals "true".

Set marked="false".

For each unmarked variable $A \in V$,

For each rule $A \to \gamma$,

· If every symbol in γ has been marked, then mark A and set marked="true".

If S is marked, then reject.

Else accept.

The above algorithm marks all variables that are capable of deriving a word $w \in \Sigma^*$. Therefore, the algorithm rejects iff S is marked, since in this case $L(G)$ will not be empty. □

Example 2.16. Apply the above algorithm to the following CFG.

$S \to ASC|Bb$

$A \to C|S$

$B \to a|\varepsilon$

$C \to b|B$

Solution.

Round 1: Mark all terminals b, a, ε

Repeat:

Round 2: Mark any variable with a rule body consisting only of marked symbols

B $(B \to a)$, C $(C \to b)$

Round 3: A $(A \to C)$, S $(S \to Bb)$

It is interesting to know that, unlike $\texttt{Equals}_{\texttt{DFA}}$

Theorem 2.17. $\texttt{Equals}_{\texttt{CFG}}$ is provably undecidable!

Exercises

1. Recall the recursive `accept` procedure described in the proof that $\texttt{Accept}_{\texttt{CFG}}$ is decidable. Draw the recursion tree that arises during the computation of `accept` on inputs G, $w = abab$, and $\nu = S_0$, where CFG G has the rules

$$S_0 \to SS \mid AB \mid WB \mid \varepsilon$$

$$S \to SS \mid AB \mid WB$$

$$W \to AS$$

$$A \to a$$

$$B \to b.$$