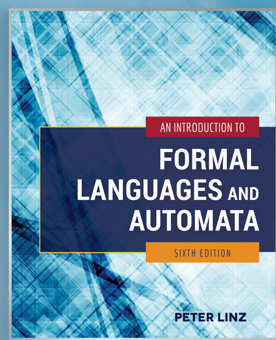
Chapter 12

LIMITS OF ALGORITHMIC COMPUTATION



© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com

Learning Objectives

At the conclusion of the chapter, the student will be able to:

- Explain and differentiate the concepts of computability and decidability
- Define the Turing machine halting problem
- Discuss the relationship between the halting problem and recursively enumerable languages
- Give examples of undecidable problems regarding Turing machines to which the halting problem can be reduced
- Give examples of undecidable problems regarding recursively enumerable languages
- Determine if there is a solution to an instance of the Post correspondence problem
- Give examples of undecidable problems regarding context-free languages

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com

Computability and Decidability

- Are there questions which are clearly and precisely stated, yet have no algorithmic solution?
- As stated in chapter 9, a function f is *computable* if there exists a Turing machine that computes the value of f for all arguments in its domain
- Since there may be a Turing machine that can compute f for part of the domain, it is crucial to define the domain of f precisely
- The concept of decidability applies to computations that result in a "yes" or "no" answer: a problem is *decidable* if there exists a Turing machine that gives the correct answer for every instance in the domain

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com

The Turing Machine Halting Problem

- The Turing machine *halting problem* can be stated as: Given the description of a Turing machine M and an input string w, does M perform a computation that eventually halts?
- The domain of the problem is the set of all Turing machines and all input strings w
- Any attempts to simulate the computation on a universal Turing machine face the problem of not knowing if/when M has entered an infinite loop
- By Theorem 12.1, there does not exist any Turing machine that finds the correct answer in all instances; <u>the halting problem is therefore undecidable</u>

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com

The Halting Problem and Recursively Enumerable Languages

- Theorem 12.2 states that, if the halting problem were decidable, then every recursively enumerable language would be recursive
- Assume that L is a recursively enumerable language and M is a Turing machine that accepts L
- If H is a Turing machine that solves the halting problem, then we can apply H to the accepting machine M
  - If H concludes that M does not halt, then by definition the input string is not in L
  - If H concludes that M halts, then M will determine if the input string is in L
- Consequently, we would have a membership algorithm for L, but we know that one does not exist for some recursively enumerable languages, therefore contradicting our assumption that H exists

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

Reducing One Undecidable Problem to Another

- A problem A is *reduced* to a problem B if the decidability of A follows from the decidability of B
- An example is the *state-entry problem*: given any Turing machine M and string w, decide whether or not the state q is ever entered when M is applied to w
- If we had an algorithm that solves the state-entry problem, it could be used to solve the halting problem
- However, because the halting problem is undecidable, the state-entry problem must also be undecidable

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

The Blank-Tape Halting Problem

- Given a Turing machine M, determine whether or not M halts if started with a blank tape
- To show that the problem is undecidable,
  - Given a machine M and input string w, construct from M a new machine M_w that starts with a blank tape, writes w on it, and acts like M
  - Clearly, M_w will halt on a blank tape if and only if M halts on w
  - If we start with M_w and apply the blank-tape halting problem algorithm to it, we would have an algorithm for the halting problem
  - Since the halting problem is known to be undecidable, the same must be true for the blank-tape version

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

The Undecidability of the Blank-Tape Halting Problem

- Figure 12.3 illustrates the process used to establish the result that the blank-tape halting problem is undecidable
- After M_w is built, the presumed blank-tape halting problem algorithm would be applied to M_w, yielding an algorithm for the halting problem, which leads to a contradiction

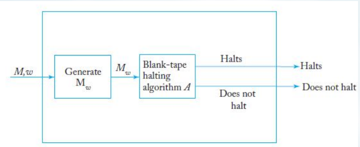


FIGURE 12.3 Algorithm for the halting problem.

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

Undecidable Problems for Recursively Enumerable Languages

- As illustrated before, there is no membership algorithm for recursively enumerable languages
- Recursively enumerable languages are so general that most related questions are undecidable
- Usually, there is a way to reduce the halting problem to questions regarding recursively enumerable languages, such as
  - Is the language generated by an unrestricted grammar empty?
  - Is the language accepted by a Turing machine finite?

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

Is the Language Generated by an Unrestricted Grammar Empty?

- Given an unrestricted grammar G, determine whether or not L(G) is empty
- To show that the problem is undecidable,
  - Given a Turing machine M and string w, modify M to create a new machine M_w, so that M_w saves its input on a special part of its tape, and whenever it enters a final state, it accepts the input only if the input is equal to w
  - Construct a grammar G_w that generates $L(M_w)$
  - Since $L(M_w) = L(M) \cap \{ w \}$, $L(G_w)$ is nonempty *iff* $w \in L(M)$
  - Assuming there is an algorithm A for deciding whether or not an arbitrary L(G) is empty, we could apply it to G_w, which would give us a membership algorithm for any recursively enumerable language
  - But this contradicts previous results that have established there is no such membership algorithm

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

The Undecidability of the "L(G) = ∅" Problem

- Figure 12.5 illustrates the process used to establish the result that the "L(G) = ∅" problem is undecidable
- After G_w is built, the presumed emptiness algorithm A would be applied to G_w, giving a membership algorithm for recursively enumerable languages, which is impossible

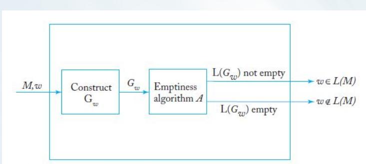


FIGURE 12.5 Membership algorithm.

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

Is the Language Accepted by a Turing Machine finite?

- Given a Turing machine M, determine whether or not L(M) is finite
- To show that the problem is undecidable,
  - Given a Turing machine M and string w, modify M to create a new machine M^, so that if any halting state of M is reached, M^ accepts all input
  - Have M^ generate w on an unused portion of its tape and perform the same computations as M, so that
    - if M halts in any configuration, then M^ halts in a final state, and
    - If M does not halt, then M^ will not halt either
  - As a result, M^ either accepts ∅ or the infinite language Σ^+
  - Assuming there is an algorithm A for deciding whether or not L(M) is finite, we could apply it to M^, which would give us a solution to the halting problem
  - But this contradicts previous results that have established that the halting problem is undecidable

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

The Undecidability of the "L(M) is Finite" Problem

- Figure 12.6 illustrates the process used to establish the result that the "L(M) is finite" question is undecidable
- After an algorithm generates M^, the presumed finiteness algorithm A would be applied to M^, resulting in a solution to the halting problem, which is impossible

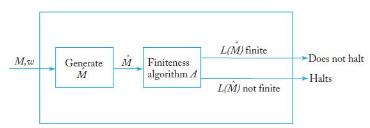


FIGURE 12.6

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com

The Post Correspondence Problem

- Given two sequences of n strings on some alphabet Σ, for instance

  $A = w_1, w_2, \ldots, w_n$ and $B = v_1, v_2, \ldots, v_n$

  there is a Post correspondence solution (PC solution) for the pair (A, B) if there is a nonempty sequence of integers i, j, …, k, such that $w_i w_j \ldots w_k = v_i v_j \ldots v_k$
- As shown in Example 12.5, assume A and B consist of

  $w_1 = 11$, $w_2, = 100$, $w_3 = 111$ and $v_1 = 111$, $v_2, = 001$, $v_3 = 11$

  A PC solution for this instance of (A, B) exists, as shown below

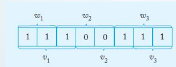


FIGURE 12.7

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com

The Undecidability of the Post Correspondence Problem

- The Post correspondence problem is to devise an algorithm that determines, for any (A, B) pair, whether or not there exists a PC solution
- For example, there is no PC solution if A and B consist of

  $w_1 = 00$, $w_2, = 001$, $w_3 = 1000$ and $v_1 = 0$, $v_2, = 11$, $v_3 = 011$
- Theorem 12.7 states that there is no algorithm to decide if a solution sequence exists under all circumstances, so the Post correspondence problem is undecidable
- Although a proof of theorem 12.7 is quite lengthy, this very important result is crucial for showing the undecidability of various problems involving context-free languages

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com

Undecidable Problems for Context-Free Languages

- The Post correspondence problem is a convenient tool to study some questions involving context-free languages
- The following questions, among others, can be shown to be undecidable
  - Given an arbitrary context-free grammar G, is G ambiguous?
  - Given arbitrary context-free grammars G_1 and G_2, is $L(G_1) \cap L(G_2) = \varnothing$?
  - Given arbitrary context-free grammars G_1 and G_2, is $L(G_1) = L(G_2)$?
  - Given arbitrary context-free grammars G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?

© Jeffrey Van Daele/ShutterStock, Inc. Copyright © 2017 by Jones & Bartlett Learning, LLC an Ascend Learning Company www.jblearning.com