Notes on Models of Computation
Chapter 12

H. Conrad Cunningham

06 April 2022

Contents

Copyright (C) 2015, 2022, H. Conrad Cunningham
Professor of Computer and Information Science
University of Mississippi
214 Weir Hall
P.O. Box 1848
University, MS 38677
(662) 915-7396 (dept. office)

Browser Advisory: The HTML version of this textbook requires a browser that supports the display of MathML. A good choice as of April 2022 is a recent version of Firefox from Mozilla.

Note: These notes were written primarily to accompany my use of Chapter 1 of the Linz textbook *An Introduction to Formal Languages and Automata* [[1].

12 Limits of Algorithmic Computation

In Linz Chapter 9, we studied the *Turing thesis*, which concerned *what Turing machines can do.*

This chapter we study: What Turing machines *cannot* do.

This chapter considers the concepts:

- computability
- decidability

12.1 Some Problems That Cannot Be Solved with Turing Machines

12.1.1 Computability

Recall the following definition from Chapter 9.

Linz Definition 9.4 (Turing Computable): A function f with domain D is said to be *Turing-computable*, or just *computable*, if there exists some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ such that

$$q_0 w \vdash_M^* q_f f(w),\ q_f \in F,$$

for all $w \in D$.

Note:

- A function f can be computable only if it is defined on the entire domain D.
- Otherwise, f is *uncomputable*.
- So the domain of f is crucial to the issue of computability.

12.1.2 Decidability

Here we work in a simplified setting: the result of a computation is either "yes" or "no". In this context, the problem is considered either decidable or undecidable.

Problem: We have a set of related statements, each either true or false.

This problem is *decidable* if and only if there exists a Turing machine that gives the correct answer for every statement in the domain. Otherwise, the problem is *undecidable*.

Example problem statement: For a context-free grammar G, the language $L(G)$ is ambiguous. This is a true statement for some G and false for others.

If we can answer this question, with either the result true or false, for every context-free grammar, then the problem is decidable. If we cannot answer the question for some context-free grammar (i.e., the Turing machine does not halt), then the problem is undecidable.

(In Linz Theorem 12.8, we see that this question is actually undecidable.)

12.1.3 The Turing Machine Halting Problem

Given the description of a Turing machine M and input string w, does M, when started in the initial configuration q_0w, perform a computation that eventually halts?

What is the domain D?

- all Turing machines and all strings w on the Turing machine's alphabet

We cannot solve this problem by simulating M. That is an infinite computation if the Turing machine does not halt.

We must analyze the Turing machine description to get an answer for any machine M and string w. *But no such algorithm exists!*

Linz Definition 12.1 (Halting Problem): Let w_M be a string that describes a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ and let w be a string in M's alphabet. Assume that w_M and w are encoded as strings of 0's and 1's (as suggested in Linz Section 10.4). A solution to the *halting problem* is a Turing machine H, which for any w_M and w, performs the computation

$$q_0 w_M w \ \vdash^* \ x_1 q_y x_2$$

if M is applied to w halts, and

$$q_0 w_M w \ \vdash^* \ y_1 q_n y_2,$$

if M is applied to w does not halt. Here q_y and q_n are both final states of H.

Linz Theorem 12.1 (Halting Problem is Undecidable): There does not exist any Turing machine H that behaves as required by Linz Definition 12.1. Thus the *halting problem is undecidable.*

Proof: Assume there exists such a Turing machine H that solves the halting problem.

The input to H is $w_M w$, where w_M is a description of Turing machine M. H must halt with a "yes" or "no" answer as indicated in Linz Figure 12.1.

We next modify H to produce a Turing machine H' with the structure shown in Linz Figure 12.2.

When H' reaches a state where H halts, it enters an infinite loop.

From H' we construct Turing machine $\hat{H}$, which takes an input w_M and copies it, ending in initial state q_0 of H'. After that, it behaves the same as H'.

The behavior of $\hat{H}$ is

$$q_0 w_M \ \vdash^*_{\hat{H}} \ q_0 w_M w_M \ \vdash^*_{\hat{H}} \ \infty$$

if M applied to w_M halts, and

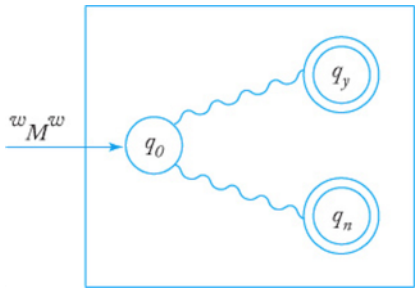

Figure 1: **Linz Fig. 12.1: Turing Machine** H

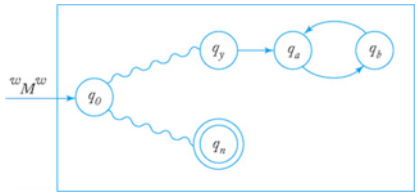

Figure 2: **Linz Fig. 12.2: Turing Machine** H'

$$q_0 w_M \vdash^*_{\hat{H}} q_0 w_M w_M \vdash^*_{\hat{H}} y_1 q_n y_2$$

if M applied to w_M does not halt.

Now $\hat{H}$ is itself a Turing machine, which can be also be encoded as a string $\hat{w}$.

So, let's apply $\hat{H}$ to its own description $\hat{w}$. The behavior is

$$q_0 \hat{w} \vdash^*_{\hat{H}} \infty$$

if $\hat{H}$ applied to $\hat{w}$ halts, and

$$q_0 \hat{w} \vdash^*_{\hat{H}} y_1 q_n y_2$$

if M applied to $\hat{w}$ does not halt.

In the first case, $\hat{H}$ goes into an infinite loop (i.e., does not halt) if $\hat{H}$ halts. In the second case, $\hat{H}$ halts if $\hat{H}$ does not halt. This is clearly impossible!

Thus we have a contradiction. Therefore, there exists no Turing machine H. The halting problem is undecidable. QED.

It may be possible to determine whether a Turing machine halts in specific cases by analyzing the machine and its input.

However, this theorem says that there exists no algorithm to solve the halting problem for all Turing machines and all possible inputs.

Linz Theorem 12.2: If the halting problem were decidable, then every recursively enumerated language would be recursive. Consequently, the halting

problem is undecidable.

Proof: Let L be a recursively enumerable language on Σ, M be a Turing machine that accepts L, and w_M be an encoding of M as a string.

Assume the halting problem is decidable and let H be a Turing machine that solves it.

Consider the following procedure.

1. Apply H to $w_M w$.
2. If H says "no", then $w \notin L$.
3. If H says "yes", then apply M to w, which will eventually tell us whether $w \in L$ or $w \notin L$.

The above is thus a membership algorithm, so L must be recursive. But we know that there are recursively enumerable languages that are not recursive. So this is a contradiction.

Therefore, H cannot exist and the halting problem is undecidable. QED.

12.1.4 Reducing One Undecidable Problem to Another

In the above, the halting problem is *reduced* to a membership algorithm for recursively enumerable languages.

A problem A is *reduced* to problem B if the decidability of B implies the decidability of A. We transform a new problem A into a problem B whose decidability is already known.

Note: The Linz textbook gives three example reductions in Section 12.1

12.2 Undecidable Problems for Recursively Enumerable Languages

Linz Theorem 12.3 (Empty Unrestricted Grammars Undecidable): Let G be an unrestricted grammar. Then the problem of determining whether or not

$$L(G) = \emptyset$$

is undecidable.

Proof: See Linz Section 12.2 for the details of this reduction argument. The decidability of membership problem for recursively enumerated languages implies the problem in this theorem.

Linz Theorem 12.4 (Finiteness of Turing Machine Languages is Undecided): Let M be a Turing Machine. Then the question of whether or not $L(M)$ is finite is undecidable.

Proof: See Linz Section 12.2 for the details of this proof.

Rice's theorem, a generalization of the above, states that any nontrivial property of a recursively enumerable language is undecidable. The adjective "nontrivial" refers to a property possessed by some but not all recursively enumerated languages.

12.3 The Post Correspondence Problem

This section is not covered in this course.

12.4 Undecidable Problems for Context-Free Languages

Linz Theorem 12.8: There exists no algorithm for deciding whether any given context-free grammar is ambiguous.

Proof: See Linz Section 12.4 for the details of this proof.

Linz Theorem 12.9: There exists no algorithm for deciding whether or not

$$L(G_1) \cap L(G_2) = \emptyset$$

for arbitrary context-free grammars G_1 and G_2.

Proof: See Linz Section 12.4 for the details of this proof.

Keep in mind that the above and other such decidability results do not eliminate the possibility that there may be specific cases–perhaps even many interesting and important cases–for which there exist decision algorithms.

However, these theorems do say that there are no general algorithms to decide these problems. There are always some cases in which specific algorithms will fail to work.

12.5 A Question of Efficiency

This section is not covered in this course.

12.6 References

[1] Peter Linz. 2011. *Formal languages and automata* (Fifth ed.). Jones & Bartlett, Burlington, Massachusetts, USA.