Chapter 9

Undecidable Problems

A Language That Can't Be Accepted, and a Problem That Can't Be Decided

- Definition 9.1: Let:
  - $NSA = \{e(T) \mid T \text{ is a TM and } e(T) \notin L(T)\}$
  - $SA = \{e(T) \mid T \text{ is a TM and } e(T) \in L(T)\}$
    - ("non-self-accepting" and "self-accepting")
- Theorem 9.2:
  - The language NSA is not recursively enumerable
  - The language SA is recursively enumerable but not recursive
  - Proof: NSA is not r.e. because of the diagonal argument: for every T, $NSA \neq L(T)$ because $e(T) \in NSA$ if and only if $e(T) \notin L(T)$

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- Proof (Cont'd.) Let E be the language $\{e(T) \mid T$ is a TM$\}$. It follows from Theorem 7.36 that E is recursive
- For every string $z \in \{0,1\}^*$, exactly one of these statements is true
  - z does not represent a TM
  - z represents a TM that accepts z
  - z represents a TM that does not accept z
- In other words
  - $\{0,1\}^* = NSA \cup SA \cup E'$
  - Because the three sets on the right are disjoint, $NSA = (SA \cup E')' = SA' \cap E$

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- If SA were recursive then SA' would be recursive, and then NSA would be recursive
- But NSA is not recursively enumerable and therefore can't be recursive
- To finish the proof we need to show that SA is recursively enumerable. An algorithm to accept SA is the following:
  - For $x \in \{0,1\}^*$, if $x \notin E$, then $x \notin SA$; if $x = e(T)$, then run T on x, and accept if and only if T accepts

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- The statement of the theorem says that there is no algorithm to determine whether a given string represents a TM that accepts its own encoding
  - It might seem that for a TM T, deciding whether T accepts the string $e(T)$ is particularly difficult, but this is not the right interpretation
  - All we needed for the diagonal argument was a string associated with T; we chose $e(T)$, but we could just as easily have used something else
- The more correct conclusion is that it's hard to answer questions about TMs and the languages they accept

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- For most TMs, it probably doesn't matter whether it accepts its own encoding; but the question is still interesting
  - Even though it is easy to state, there is no algorithm to answer it. There are other precise, easy-to-formulate questions for which it *would* be useful to have algorithms but for which there are no such algorithms

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- In discussing a general decision problem P, we start by choosing an encoding function e that allows us to represent instances I by strings $e(I)$ over some alphabet Σ
- We give the names $Y(P)$ and $N(P)$ to the sets of strings in Σ^* that represent yes- and no-instances
  - If $E(P) = Y(P) \cup N(P)$, then we have a third subset $E(P)'$ of strings in Σ^* that don't represent instances of P

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- The requirements on *e* are straightforward
  - It must be 1-1, so that a string can't represent more than one instance
  - There must be an algorithm to decide whether a given string in Σ^* is $e(I)$ for some instance I of P
  - If it is of the form $e(I)$, it must be possible to decode it and recover I
- We refer to a function e satisfying these requirements as a *reasonable* encoding of instances of P

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- Definition 9.3: If P is a decision problem, and e is a reasonable encoding of instances of P over the alphabet Σ, we say that P is *decidable* if $Y(P) = \{e(I) \mid I \text{ is a yes-instance of } P\}$ is a recursive language
- Theorem 9.4: The decision problem *Self-Accepting* is undecidable
- Proof: Because $SA = Y$ (*Self-Accepting*), the statement follows immediately from Theorem 9.2

A Language That Can't Be Accepted, and a Problem That Can't Be Decided (cont'd.)

- Theorem 9.5: For every decision problem P, P is decidable if and only if the complementary problem P' is decidable
- Proof: for exactly the same reasons as in Theorem 9.2, we have $N(P) = Y(P)' \cap E(P)$
  - $E(P)$ is assumed to be recursive
  - If $Y(P)$ is recursive then so is $Y(P)'$, and therefore so is $N(P) = Y(P')$
  - The other direction is similar

Reductions and the Halting Problem

- We can often solve problems by reducing them to other, simpler ones
- In this section, we consider reducing one decision problem to another
- The two crucial features in a reduction are:
  - For every instance I of the problem we start with, we must be able to obtain an instance $F(I)$ of the second problem algorithmically
  - The answer to the second question for the instance $F(I)$ must be the same as the answer to the original question for I

Reductions and the Halting Problem (cont'd.)

- Definition 9.6:
  - Suppose P_1 and P_2 are decision problems
    - We say P_1 is reducible to P_2 $(P_1 \leq P_2)$ if there is an algorithm that finds, for an arbitrary instance I of P_1, an instance $F(I)$ of P_2 such that the two answers (the answer to P_1 for the instance I, and the answer to P_2 for the instance $F(I)$) are the same
  - If L_1 and L_2 are languages over alphabets Σ_1 and Σ_2
    - We say L_1 is reducible to L_2 $(L_1 \leq L_2)$ if there is a Turing-computable function $f: \Sigma_1^* \rightarrow \Sigma_2^*$ such that for every $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$

Reductions and the Halting Problem (cont'd.)

- Theorem 9.7:
  - Suppose $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, and $L_1 \leq L_2$
    - If L_2 is recursive, then L_1 is recursive (Proof: we can decide whether a string is in L_1 by using the reduction and deciding whether the resulting string is in L_2)
  - Suppose P_1 and P_2 are decision problems, and $P_1 \leq P_2$
    - If P_2 is decidable, then P_1 is decidable (Intuitive proof: we can decide whether an instance of P_1 is a yes-instance by using the reduction and deciding whether the resulting instance of P_2 is a yes-instance)
  - We will be interested in the contrapositive statement: If P_1 is undecidable, then P_2 is also.

Reductions and the Halting Problem (cont'd.)

- A slightly more careful proof of the second part:
  - Suppose that F is an algorithm reducing P_1 to P_2, and that we have reasonable encoding functions e_1 and e_2 for the instances of P_1 and P_2
  - Assume $Y(P_2)$ is recursive
  - We need to show $Y(P_1)$ is also recursive
  - For an arbitrary string x, we can decide whether x represents an instance of P_1
  - If not, then $x \notin Y(P_1)$

Reductions and the Halting Problem (cont'd.)

- A slightly more careful proof of the second part (cont'd.)
  - If $x \in E(P_1)$, then x represents an instance I of P_1, and we have this sequence of equivalences
    - $x \in Y(P_1) \Leftrightarrow I$ is a yes-instance of P_1

      $\Leftrightarrow F(I)$ is a yes-instance of P_2

      $\Leftrightarrow e_2(F(I)) \in Y(P_2)$
  - Because $Y(P_2)$ is recursive, we have an algorithm to decide whether $e_2(F(I)) \in Y(P_2)$ and so we have an algorithm to decide whether $x \in Y(P_1)$

Reductions and the Halting Problem (cont'd.)

- Consider two more decision problems:
  - *Accepts*: Given a TM T and a string w, is $w \in L(T)$?
  - *Halts*: Given a TM T and a string w, does T halt (either by accepting or by rejecting) on input w? (This is called the *halting problem*)

Reductions and the Halting Problem (cont'd.)

- Theorem 9.8: Both *Accepts* and *Halts* are undecidable
- For the first statement, we just need to show that *Self-accepting* $\leq$ *Accepts*
  - A reduction from *Self-accepting* to *Accepts* is $F(T) = (T, e(T))$
  - We can compute this algorithmically.
- For the second statement, we can reduce *Accepts* to *Halts* (see the book for the details). *Accepts* is undecidable; therefore, *Halts* is undecidable

More Decision Problems Involving Turing Machines

- Theorem 9.9: The following five decision problems are undecidable:
  1. *Accepts*-Λ: Given a TM T, is $\Lambda \in L(T)$?
  2. *AcceptsEverything*: Given a TM T with input alphabet Σ, is $L(T) = \Sigma^*$?
  3. *Subset*: Given two TMs T_1 and T_2, is $L(T_1) \subseteq L(T_2)$?
  4. *Equivalent*: Given two TMs T_1 and T_2, is $L(T_1) = L(T_2)$?
  5. *WritesSymbol*: Given a TM T and a symbol a in the tape alphabet of T, does T ever write an a if it starts with an empty tape?

More Decision Problems Involving Turing Machines (cont'd.)

- Proof: In each case, we construct a reduction

  1. $Accepts \leq Accepts\text{-}\Lambda$
  2. $Accepts\text{-}\Lambda \leq AcceptsEverything$
  3. $AcceptsEverything \leq Subset$
  4. $Subset \leq Equivalent$
  5. $Accepts\text{-}\Lambda \leq WriteSymbol$

  For details of the reductions, see the book

More Decision Problems Involving Turing Machines (cont'd.)

- Theorem 9.10:
  - The decision problem *WritesNonblank* is decidable, where *WritesNonblank* is the problem: Given a TM *T* with *n* nonhalting states, does *T* ever write a nonblank symbol on its tape, if it starts with a blank tape?
- Proof sketch:
  - An algorithm to decide *WritesNonblank* is to trace *T* for *n* moves, or until it halts, whichever comes first
  - If by that time no nonblank symbol has been written, none ever will be

More Decision Problems Involving Turing Machines (cont'd.)

- Definition 9.11:
  - A property R of Turing machines is called a *language property* if, for every Turing machine T having property R, and every other TM T_1 with $L(T_1) = L(T)$, T_1 also has property R
  - A language property of TMs is *nontrivial* if there is at least one TM that has the property and at least one that doesn't
- Some properties are clearly language properties:
  - "Accepts at least two strings"
  - "Accepts Λ"

More Decision Problems Involving Turing Machines (cont'd.)

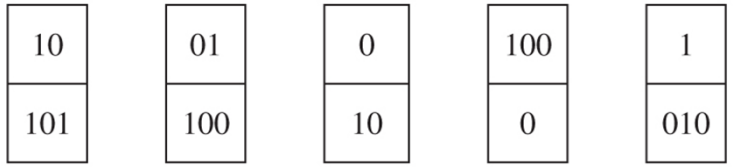
- Some properties are not language properties:
  - "Writes a nonblank symbol when started on an empty tape"
- Theorem 9.12, Rice's Theorem:
  - If R is a nontrivial language property of TMs, then the decision problem P_R : Given a TM T, does T have property R? is undecidable
  - Idea of the proof: reduce *Accepts*-Λ either to $P_{not\text{-}R}$ or to P_R, depending on whether the empty language has property R

More Decision Problems Involving Turing Machines (cont'd.)

- It's now clear that it's difficult to answer questions about Turing machines and the strings they accept
- A few more undecidable problems about a TM T:
  1. (For some language L) *AcceptsL*: Given a TM T, is $L(T) = L$?
  2. *AcceptsSomething*: Is there at least one string in $L(T)$?
  3. *AcceptsTwoOrMore*: Does $L(T)$ have at least two elements?
  4. *AcceptsFinite*: is $L(T)$ finite?
  5. *AcceptsRecursive*: is $L(T)$ recursive?

Post's Correspondence Problem

- Here's a simple case: Think of each of the five rectangles in figure (a) as a domino
  - You have an unlimited supply of each
  - Can we arrange them (with duplicates allowed) so that the top row of symbols matches the bottom row?
  - In this example, the answer is yes, as we see in figure (b)
  - In general, this is Post's Correspondence Problem



(a)

10	1	01	0	100	100	0	100
101	010	100	10	0	0	10	0

(b)

Post's Correspondence Problem (cont'd.)

- Definition 9.14:
  - An instance of Post's correspondence problem (PCP) is a set $\{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)\}$ of pairs, where $n \geq 1$ and each α_i and each β_i is a nonnull string over an alphabet Σ.
  - The decision problem: Given an instance of this type, do there exist a positive integer k and sequence of integers $i_1, i_2, \ldots, i_k$ with each i_j satisfying $1 \leq i_j \leq n$, satisfying $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$?
  - An instance of the *modified* PCP (MPCP) adds the requirement that i_1 be 1

Post's Correspondence Problem (cont'd.)

- Definition 9.14: (cont'd.)
  - Instances of *PCP* and *MPCP* are called *correspondence systems* and *modified correspondence systems*.
  - For an instance of either type, if it is a yes-instance we will say that there is a *match* for the instance
  - We show that $Accepts \leq MPCP \leq PCP$ and deduce that both problems are undecidable (and that neither is completely unrelated to TMs after all)

Post's Correspondence Problem (cont'd.)

- Theorem 9.15: $MPCP \leq PCP$
  - Proof: by construction; see book
- Theorem 9.16: $Accepts \leq MPCP$
  - Proof: By constructing an instance of MPCP in which the strings involve TM configurations; see book
- Theorem 9.17: Post's correspondence problem is undecidable
  - Proof: An immediate consequence of Theorems 9.16 and 9.16

Undecidable Problems Involving Context-Free Languages

- Theorem 9.20: These two problems are undecidable
  - *CFGNonemptyIntersection*: Given two CFGs G_1 and G_2, is $L(G_1) \cap L(G_2)$ nonempty?
  - *IsAmbiguous*: Given a CFG G, is G ambiguous?
- The proof of both parts is to reduce PCP to the given problem
  - See book for details

Undecidable Problems Involving Context-Free Languages (cont'd.)

- Definition 9.21:
  - Let $T = (Q, \Sigma, \Gamma, q_0, \delta)$ be a TM
  - A *valid computation* of T is a string of the form $z_0\#z^r_1\#z_2\ldots\#z^r_{n-1}\#z_n\#$ if n is even, or $z_0\#z^r_1\#z_2\ldots\#z_{n-1}\#z^r_n\#$ if n is odd, where:
    - In either case, # is a symbol not in Γ
    - Strings z_i represent successive configurations of T on some input string x, starting with the initial configuration z_0 and ending with an accepting configuration
  - The set of valid computations of T will be denoted by C_T

Undecidable Problems Involving Context-Free Languages (cont'd.)

- Theorem 9.22: For a TM $T=(Q, \Sigma, \Gamma, q_0, \delta)$, the set C_T of valid computations of T is the intersection of two context-free languages, and its complement C_T' is a context-free language
  - Proof: See book
- Theorem 9.23: The decision problem *CFGGeneratesAll* (Given a CFG G with terminal alphabet Σ, is $L(G) = \Sigma^*$?) is undecidable
  - Idea of proof: reduce *AcceptsNothing* to *CFGGeneratesAll*, using Theorem 9.22