More Undecidability

If the languages and sets we want to examine are not index-sets, the techniques developed before the Recursion Theorem to prove them non-recursive or non-recursively-enumerable are limited: we have to prove the non-existence of functions with certain properties. The only effective tools are diagonalization and reducibility, and most of the reducibility results hinge on Rice’s Theorem, and thus on the sets under consideration being index-sets.

We now have another line of attack: the Recursion Theorem. We present a smorgasbord of undecidable problems, mostly for sets that are not index-sets, and look at a few of the proofs.

More Undecidability

b. Given two DTMs $M_1$ and $M_2$, determine whether they are equivalent (= compute the same function).

**Proof:** see textbook. $	ext{Emp}$ is reduced to $\{(x, y) \mid W_x = W_y\}$. But this is what we want if equivalence means that $M_1$ and $M_2$ compute the same function. Let $\phi_1, \phi_2$ be the functions corresponding to $M_1, M_2$. If they compute the same function, $\phi_1 = \phi_2$. Let $\phi_1$ be an everywhere undefined function (they are all identical, but can correspond to different strings), and define $g(x) = (x, \phi_1)$. Then $z \in \text{Emp} \Leftrightarrow \exists x, \phi_1, \phi_2 \Leftrightarrow g(x) = (x, \phi_2) \in A_w$, and the reduction follows.

More Undecidability

We first turn to some variants of the halting problem (undecidable problems):

a. Given a DTM $M$ and a string $y$ (in binary representation), determine whether $M$ halts on some input $z \geq y$.

**Proof:** Let $A_y = \{(x, y) \mid M_y$ halts on some input $z \geq y\}$. We prove it is not recursive by reducing $\text{Emp}$ to it (recall that $\text{Emp} = \{a \mid W_a = 0\}$).

Let $g(x) = (x, 0)$, which is clearly recursive $(\{x, 0\}$ is the standard pairing function).

- If $x \in \text{Emp}$, $M_y$ halts on some input $z \geq 0$, and $g(x) = (x, 0) \in A_y$.
- If $x \notin \text{Emp}$ (i.e., $x \in \text{Emp}$), then $M_y$ does not halt on any $y \geq 0$, and so $(x, 0) \notin A_y$; $g(x)$ is a reduction function and $E \leq_m A_y$.

More Undecidability

c. Given a DTM $M$, an input $y$, and a state $q_i$ of $M$, determine whether $M$ ever enters the state $q_i$ in the computation with input $y$.

**Proof:** Let $A_\ell = \{(x, y) \mid x, y \in \{0, 1\}^*$ and $M_y$ enters state $q_i$ in its computation with input $y\}$. We reduce the halting problem $K_\ell = \{(x, y) \mid \delta(y, q_i) \rightarrow^{*} \rightarrow \phi\}$ to $A_\ell$. Let $\{q_0, \ldots, q_n\}$ be the states of $M$. We first construct a machine $M_y$ from $M_y$, using a primitive recursive function $f(x) = z$. $M_y$ has states $\{q_0, q_1, \ldots, q_n\}$ - the states of $M_y$ to which we have added a new state $q_{n+1}$, the halt state of $M_y$. We go from $x$ to $z$ by adding all the instructions of the form $\delta(x, a) = (q_{a+1}, a, R)$ for every $a \in \Gamma; M_y$ halts on $y \Rightarrow M_y$ enters state $q_{n+1}$.

$g(x, y) = (f(x), y, \maxstates(x))$ is a reduction, since it is recursive, $g : \{0, 1\}^* \rightarrow \{0, 1\}^*$ and $z \in K_\ell \iff g(z) \in A_y$.

Note: $K_\ell$ maps, via $g$, to a subset of $A_y$. The fact that a machine that does not halt could very well enter many other states during the computation is irrelevant to the reduction.
More Undecidability

d. Given a DTM \( M \), determine whether the computation of \( M(111) \) contains a configuration in which the tape contains a substring \( 000 \).

\textbf{Pf.:} See the textbook. An interesting twist is the use of the Church-Turing Thesis to claim both existence and effective constructability for a Turing Machine with certain properties...

More generally, this result tells us that the question: "given an input, does any intermediate step of the computation contain a given pattern on the tape?" is undecidable.

More Undecidability

Two questions arise, and both have to be answered:

1. How do we represent \((G_M', x)\) as a string in \( \{0, 1\}^* \) assuming \( G_M' \) and \( x \) are represented as "canonical" strings? This is easy: use a pairing function, which is recursive.

2. How do we represent \( G_M' \) as a string in \( \{0, 1\}^* \)? We must start from \( M \), and, in primitive recursive (or, at least, provably recursive) ways, end up with a string representing all the productions of \( G_M' \). \textbf{Thms. 4.19 and 4.20} give a way of representing the productions of the grammar associated with a Turing machine in terms of an alphabet (and, therefore, uniquely, as binary strings), and the whole grammar as a concatenation of binary strings. Since various previous results on "binary string surgery" lead us to believe all this is doable in a primitive recursive manner, we will wave our hands a bit and declare the whole thing valid: another undecidable question.

More Undecidability

Undecidable problems for phrase structure grammars.

a. Given a grammar \( G \) and a string \( x \), determine whether \( x \in L(G) \).

\textbf{Pf.:} \textbf{Theorem 4.20} states: If a language \( L \subseteq \{0, 1\}^* \) is Turing-acceptable, then \( L = L(G) \) for some grammar \( G \).

Let \( A_i = \{(G, x) \mid x \in L(G)\} \). We will reduce (again) the halting problem \( K_0 = \{(M, x) \mid M \text{ halts on } x\} \) to \( A_i \). (Note the confusion - more or less intentional - between pairs of (TM, input string), and parameters of a pairing function giving a single number.) Given \( M \), \textbf{Thm. 4.20} gives a grammar \( G_M' \). Define \( f(M, x) = (G_M', x) \). This is, clearly, the desired reduction, in the sense that \( (M, x) \in K_0 \iff (G_M', x) \in A_i \). One must convince oneself that \( f \) is recursive (= Turing computable & total).

Recall that \( f : \{0, 1\}^* \rightarrow \{0, 1\}^* \).

More Undecidability

b. Given a grammar \( G \) and two strings \( x \) and \( y \), determine whether \( x \Rightarrow_G y \).

\textbf{Pf.:} this is similar to the previous one, where we define the map \( f(M, x) = (G_M', S, x) \) from \( K_0 \) to \( \{G, x, y \mid x \Rightarrow y\} \). Again, some gymnastics with Turing machines, binary representations and pairing functions is necessary for the right-hand-side triple.

c. Let a grammar \( G \) be given, with \( x, y \in L(G) \). Determine whether there is a derivation of \( x \) that is longer than the shortest derivation of \( y \).

\textbf{d.} Given a grammar \( G \), determine whether \( L(G) = \emptyset \).

\textbf{Pf.:} let \( A_i = \{G \mid L(G) = \emptyset\} \). \( f(M) = G_M' \) is a reduction from \( \text{EMP} \) to \( A_i \).
More Undecidability

e. For grammars $G_1$ and $G_2$, determine whether $L(G_1) \subseteq L(G_2)$.

**Proof:** Let $G_1$ be the grammar with the single rule $S \rightarrow S \in \Sigma$. Consider the reduction $f(G) = (G, G_0) : A_0 = \{ G \mid L(G) = \emptyset \} \rightarrow A_1 = \{ (G, G_0) \mid L(G) \subseteq L(G_2) \} \subseteq \{ (G, G_2) \mid L(G) \subseteq L(G_2) \}$.

f. Let $G$ be a grammar. Determine whether $L(G)$ is context-free.

**Proof:** Let $C = \{ x \mid W_x$ is a context-free language $\}$. It is a nontrivial set-index set, since $W_n = \mathcal{X}_n(x) = \mathcal{X}_n(y)$, and both context-free and non-context-free languages exist. By Rice’s theorem it is undecidable. The function $f(x) = G_{cf}$ is a reduction $C \rightarrow \{ G \mid L(G)$ is context-free $\}$

More Undecidability

b. If $G$ is a CFG, the problem of determining whether $L(G) = \emptyset$ is decidable.

**Proof:** Clearly, $L(G) \neq \emptyset \Rightarrow \exists$ generates some string of terminals. How can we show this? Back up from the productions that derive only terminals until you find the start symbol or stop. If you don’t find it, $L(G) = \emptyset$, otherwise $L(G) \neq \emptyset$. Algorithm (H&U, ’79, p. 89):

1. OLDV := \emptyset
2. NEWV := \{ $A \rightarrow w$ for some $w \in \Sigma^*$ \}
3. while OLDV \neq NEWV do
   4. \{ OLDV := NEWV
   5. NEWV := OLDV \cup \{ $A \rightarrow \alpha$ for some $\alpha \in (\Sigma \cup OLDV)^*$ \}
   6. V’ := NEWV

More Undecidability

More Undecidability

More Undecidability

Some Decidable Problems for CFGs

a. Given a context-free grammar $G$ and a string $x$, determine whether $x \in L(G)$.

**Proof:** This depends on the Greibach Normal form Theorem: Any CF language $L$ without $\varepsilon$ can be generated by a grammar for which every production is of the form $A \rightarrow ax$, where $a \in \Sigma$ and $\alpha$ is a (possibly empty) string of non-terminals. We observed earlier in the course that any CF grammar can be replaced by an equivalent one where only the start symbol can generate $\varepsilon$, so the requirement above is not restrictive.

All we need to do is to apply, non-deterministically, $L$ derivations rules, starting from the start symbol. If the grammar has a maximum of $n$ rules for each non-terminal, a total application of $n^n$ derivations will either produce the string or fail.

More Undecidability

Claim: the algorithm collects all the nonterminals that lead to terminal strings.

**Proof of Claim.** By induction on the length of the derivation.

Length = 1: $A \rightarrow w$ is a production, and NEWV satisfies the condition - or is empty - and $A$ is added to NEWV.

Length = $n + 1$: let $A \rightarrow X_1X_2 \ldots X_n \rightarrow^* w$ by a derivation of $n + 1$ steps, where each $X_i$ is either a terminal or a non-terminal leading to $w \in \Sigma^*$ via a derivation of no more than $n$ steps. By the induction hypothesis, the non-terminal $X_i$’s are eventually added to NEWV. At the while-loop test (line 3), right after the $X_i$’s are added to NEWV, we cannot have OLDV = NEWV; thus the loop is entered and $A$ will be added to NEWV in line 5. $A$ satisfies the condition. At the end, any $A \notin V$ cannot derive a string of terminals. $L(G) \neq \emptyset \Rightarrow S \in V'$.
More Undecidability

c. If $G$ is a CFG, the problem of determining whether $L(G)$ is finite is
decidable.
d. If $G$ is a CFG, the problem of determining whether $L(G)$ is infinite is
decidable.
Pf.: construct the "derivation graph", after elimination of $\epsilon$-productions
and useless productions. If it has no cycles, $L(G)$ is finite, otherwise it
is infinite.

Some Undecidable
Problems for
CFGs

Theorem 5.43. If $G$ is a CFG, the problem of determining whether
$L(G) = \{0, 1\}^*$ is undecidable.
Proof. This proceeds by constructing a reduction from EMP. The proof
is quite technical and will be omitted for now.

Corollary 5.44. Let $G_1$ and $G_2$ be two CFGs.
a. The problem of determining whether $L(G_1) \subseteq L(G_2)$ is undecidable.
b. The problem of determining whether $L(G_1) = L(G_2)$ is undecidable.
Proof. The proof, in both cases, is based on a reduction. Let $G_0$ be a
CFG s.t. $L(G_0) = \{0, 1\}^*$. The function $f(G) = (G_0, G)$ is a reduction from
the problem of determining whether $L(G) = \{0, 1\}^*$ to either of the two
problems above.

The Post Correspondence Problem. Given a finite set of ordered
pairs $(x_1, y_1), \ldots, (x_n, y_n)$ of strings over an alphabet $\Sigma$, determine
whether there is a finite sequence of integers $i_1, \ldots, i_m$ with each
$i_j \in \{1, \ldots, n\}$, such that $x_{i_1} x_{i_2} \cdots x_{i_m} = y_{i_1} y_{i_2} \cdots y_{i_m}$.
(Repetitions are allowed.)
Ex.: 

<table>
<thead>
<tr>
<th>$x_1 = aa$</th>
<th>$x_1 = ba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1 = a$</td>
<td>$y_2 = ba$</td>
</tr>
<tr>
<td>$x_2 = ba$</td>
<td>$y_1 = a$</td>
</tr>
<tr>
<td>$y_2 = ba$</td>
<td>$y_1 = a$</td>
</tr>
</tbody>
</table>

$i_1 = 2, i_2 = 1, i_3 = 1.$

Theorem 5.45. The Post Correspondence Problem is undecidable
(with respect to some alphabet).
Proof.
More Undecidability

Corollary 5.46. The problem of determining whether two given context-free grammars $G_i$ and $G_j$ satisfy $L(G_i) \cap L(G_j) = \emptyset$ is undecidable.

Proof. Let $A = \{(G_i, G_j) | L(G_i) \cap L(G_j) = \emptyset\}$. We reduce PCP to $\lambda$. We need to construct a recursive function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that $x \in \text{PCP} \iff f(x) \in \lambda$. As discussed before, the transition from arbitrary strings over an arbitrary alphabet to binary strings can be carried out via a recursive function; context-free grammars can be encoded as binary strings, and pairs (or $n$-tuples of pairs) of strings over an alphabet can be encoded into binary via pairing functions. So, after a lot of handwaving, the idea of such a reduction becomes legitimate.

Let $\Sigma$ be a fixed alphabet with respect to which PCP is known to be undecidable. Let $\$ \in \Sigma$. For any instance $(x_1, y_1), \ldots, (x_n, y_n)$ of PCP,

$S \rightarrow S_1 \mid S_2,$

$S_1 \rightarrow xS_1y \mid \lambda x,y \in \Sigma,$

$S_2 \rightarrow yS_2x \mid \lambda x,y \in \Sigma.$

The construction is a reduction from PCP to $\lambda$, and we are done.

More Undecidability

Corollary 5.47. The problem of determining whether a given CFG $G$ is ambiguous is undecidable.

Proof. By reduction of PCP to the ambiguity problem. Let AMB denote the set of ambiguous CFGs over an alphabet $\Sigma$.

We must find a recursive function $f : \text{PCP} \rightarrow \text{AMB}$, satisfying $x \in \text{PCP} \iff f(x) \in \text{AMB}$.

1. Start with an instance $(x_1, y_1), (x_1, y_2), \ldots, (x_n, y_n)$ of PCP.
2. Define the CFG: $S \rightarrow S_1 \mid S_2,$

$S_1 \rightarrow xS_1y \mid \lambda x,y \in \Sigma,$

$S_2 \rightarrow yS_2x \mid \lambda x,y \in \Sigma.$

where $x, y \in \Sigma$. If this instance has a solution $x_1, x_2, \ldots, x_n$ (with matching $y_1, y_2, \ldots, y_n$), then the string $z = x_1x_2 \cdots x_nba^*ba^*b$ has two distinct leftmost derivations, one from $S_1$ and one from $S_2$. Thus $x \in \text{PCP} \iff f(x) \in \text{AMB}$.

More Undecidability

3. We must show that the function we constructed - the construction of the CFG grammar from the instance of the PCP - is recursive: we start with a finite set of strings over an alphabet $\Sigma$, and end with a finite set of rules over an extended alphabet. The construction can be carried out via TM, and so $f$ is recursive. See Theorems 4.19 and 4.20.

4. We now must show that $z = f(x) \in \text{AMB} \iff x \in \text{PCP}$. The assumption implies that $z$ has at least two distinct leftmost derivations, from a grammar as defined in 2 above. We can observe that, if $S_1 \Rightarrow z$, then $z$ must have the form $z = x_1x_2 \cdots x_nba^*ba^*b$, and is obtained through a unique leftmost derivation; if $S_2 \Rightarrow z$, then $z$ must have the form $z = y_1y_2 \cdots y_nba^*ba^*b$, also from a unique leftmost derivation. Since derivations from $S_1$ and $S_2$ are unique (by construction of the grammar), we must conclude that the two derivations are $S \Rightarrow S_1 \Rightarrow z$ and $S \Rightarrow S_2 \Rightarrow z$, with $z$ having both forms above.
More Undecidability

The fact that \( a, b \not\in \Sigma \), while \( x_i, y_j \in \Sigma \) \( \forall m, n \), implies that the suffixes \( ha^{m}b \ldots ba^{k}bha^{m}b \) and \( ha^{n}b \ldots ba^{l}bha^{n}b \) must match. Which, in turn, implies that \( j_1 = i_1, j_2 = i_2, \ldots \) and \( k = l \), giving us that \( x_{i_1}x_{i_2} \ldots x_{i_k} = y_{j_1}y_{j_2} \ldots y_{j_k} \). And this is the same as saying that \( \{(x_{i_1}, y_{j_1}), \ldots, (x_{i_k}, y_{j_k})\} \) is an instance of PCP.