Decidability and CFL's

We discovered earlier that all interesting (extensional and non-trivial) problems about WHILE languages were undecidable, whereas similar problems involving regular languages were decidable. In this section we discover that the situation for CFL's is a bit more complex, with some problems decidable and others undecidable.

Section 3.6 of L&P shows that the set $A_{CFL} = \{ \langle G,w \rangle \mid w \text{ in } L(G) \}$ is decidable. That is, there is there is an algorithm which, given G and w, can determine whether G generates w. We will discuss this algorithm on Wednesday if there is sufficient time.

Another decidable problem for cfl's is $E_{CFL} = \{ G \mid L(G) = \emptyset \}$. A fairly efficient algorithm is the following:

Mark all terminal symbols of G;
Repeat until no new variables get marked:
Mark variable A if G has a rule $A \rightarrow U_1 \ldots U_n$ where all of the U_i are marked.
If S is not marked than $L(G) = \emptyset$, else $L(G) \neq \emptyset$.

[The basic idea is that all marked symbols can generate a string of terminal symbols. Thus if S is marked then it can generate a string of terminals, and hence $L(G) \neq \emptyset$.]

However, there are a number of problems for cfl's that are not decidable. One is given in Theorem 5.5.2a of the text: $ALL_{CFL} = \{ G \mid L(G) = \Sigma^* \}$ is not decidable.

As usual we will show that if ALL_{CFL} is decidable then we can solve the halting problem for Turing machines. Because we showed last time that WHILE-programs can simulate Turing machines and vice-versa, we know that the halting problem for Turing machines is also undecidable. The idea behind the proof is that, given <M,w>, we can define a context-free language L that represents the collection of all strings that do not represent halting computations of M on input w. Thus M halts on w iff $L \neq \Sigma^*$.

To do this, we need a way to represent all computations of M in Σ^*. Previously we wrote configurations in the form $(q,L\underline{a}R)$. To avoid writing underlines on characters, we will now write the same configuration as LqaR. That is, we write the state just to the left of the character currently being scanned. We presume that the language of M does not include any of the symbols used to represent states or the character #.

We will write a computation as $\#C_1\#C_2\#\ldots\#C_n\#$, where C_i is the configuration just before the i^{th} step of the computation.

We want to characterize those strings that do *not* correspond to accepting computations of M on w. A string fails to be an accepting computation if

1. It does not start with #sBw# for s the start state of M.
2. It doesn't end with $\#C_n\#$ where C_n is a halting configuration (i.e., has state h in H).
3. Some C_i does not properly yield C_{i+1} under the rules of M.

Note that if M does not halt on w then all strings fail to be accepting computations. Let L be the set of strings that fail to be accepting computations of M on w. Then L = Σ^* iff M does not halt on w.

We claim that L is a cfl. Rather than describing L with a grammar, we will instead describe a pda that accepts L. Because we have an algorithm to convert pda's to grammars (Lemma 3.4.2, whose proof was skipped in class), we can construct a cfg, G, such that determining whether G in ALL_{CFG} would solve the halting problem. I.e., if G is the grammar generating the language L, then G in ALL_{CFG} iff M does not halt on w.

Now all we have to do is to construct a pda, $P_{M,w}$, that accepts L. The pda begins by non-deterministically guessing which of the three conditions given above fails.

1. If it guesses the first, then it checks to make sure the input does not start with #sBw#. It halts and accepts if it does not start with #sBw#.
2. If it guesses the second, then it non-deterministically guesses where the last configuration starts and makes sure that it is not a legal final configuration (i.e., that it does not have a single state that is a halting state). If it succeeds then it halts and accepts.
3. If it guesses the third condition, then it non-deterministically guesses the i such that $C_i \vdash C_{i+1}$ fails. When it reads C_i, it copies it onto the stack one character at a time until it reaches the next #. It then reads C_{i+1}, comparing characters looking either for a mismatch with C_i or for an incorrect transition. We can detect incorrect transitions as follows: If C_i is LaqbR and δ(q,b) = (r,d, □) then C_{i+1} should be LradR, while if δ(q,b) = (r,d, □) then C_{i+1} should be LadrR. [Of course we are trying to make sure that either the characters to the left or right *don't* match or the transition is not represented accurately.]

While it is somewhat tedious to write out the details of the pda transitions, they are straightforward except for one problem that we brushed over above: When C_i is pushed onto the stack, it will then be popped off backwards – i.e., from right to left rather than left to right. Thus we cannot easily compare C_i with C_{i+1}!

However, it is easy to overcome this problem by making a slightly different definition of computation. We simply write all configurations for even steps backwards! That is we write $\#C_1\#C_2^{rev}\#C_3\#C_4^{rev}\#\ldots\#C_n'\#$ where C_n' is reversed only if n is even. Now there is no difficulty in comparing consecutive configurations, though the details will be slightly different in going from odd to even than going from even to odd.

Theorem 5.5.2: The following problems are undecidable:

(a) Given a context-free grammar, G, is L(G) = Σ^* ? (I.e., the set ALL_{CFL} = { G | L(G) = Σ^*} is undecidable.)
(b) Given two context-free grammars, G_1 and G_2, is $L(G_1) \supseteq L(G_2)$? (I.e., the set SUP_{CFL} = { <G_1, G_2> | $L(G_1) \supseteq L(G_2)$} is undecidable.)
(c) Given two context-free grammars, G_1 and G_2, is $L(G_1) = L(G_2)$? (I.e., the set EQ_{CFL} = { <G_1, G_2> | $L(G_1) = L(G_2)$} is undecidable.)

Proof: Part (a) was proved above. Show that if (b) is decidable, then (a) would be as well. Let G' be a grammar generating Σ^*. To determine if, given G, whether $L(G) = \Sigma^*$ just ask the algorithm for (b) if $L(G) \supseteq L(G')$? If so then $L(G) = \Sigma^*$. The same proof works for part (c).

There are also other important problems involving cfl's that are undecidable:

Theorem: The following problems are undecidable:

(a) Given two context-free grammars, G_1 and G_2, does $L(G_1) \cap L(G_2) = \emptyset$? (I.e., the set $EINT_{CFL} = \{ <G_1, G_2> \mid L(G_1) \cap L(G_2) = \emptyset\}$ is undecidable.)

(d) Given a context-free grammar, G, is G ambiguous? (I.e., the set $AMB_{CFL} = \{ G \mid L(G) \text{ is ambiguous}\}$ is undecidable.)

(b)

The proofs of both of these parts depends on first proving the Post Correspondence Problem (see problem 5.5.2) is undecidable, so we will skip the proofs here.