ECS120 Introduction to the Theory of Computation
Fall Quarter 2007

Discussion 7 Notes
Wednesday November 14, 2007

Decision Procedures

A **decision procedure** is a procedure that provides a yes or no answer for a decision problem. A **decider** is a Turing machine that halts on all inputs. (It always accepts or rejects a string.) Therefore, we can re-formulate our decision procedures as deciders.

Consider the procedure for deciding if $L(G) = \emptyset$ for a context-free grammar G. The general idea behind this is to:

1. Mark all the terminal symbols.
2. Until there is nothing more to mark:
   (a) If the right-hand side (RHS) is completely marked, mark the left-hand side (LHS).
   (b) Mark every occurrence of that LHS.
3. If the start symbol is marked, then output `YES`. Otherwise output `NO`.

Consider the grammar G for $a*b*$:

$$\begin{array}{rcl} S & \rightarrow & A|B|AB \\ A & \rightarrow & aA|\epsilon \\ B & \rightarrow & bB|\epsilon \end{array}$$

To create a Turing machine to decide this, we need to somehow input our grammar to the Turing machine as a string. We can create a string from the formal description of the grammar. For the grammar above, our input tape might look like this:

S	A	B	#	a	b	ϵ	#	S	A	#	S	B	#	S	A	B	#	

(continued on next line...)

A	a	A	#	A	ϵ	#	B	b	B	#	B	ϵ	#	␣	$\cdots$

We know that the first symbol is the start symbol, followed by variables. After the first # we have our alphabet symbols. Following the next # we have rules where the first symbol is the left hand side, and all the symbols until the next # make up the right hand side. We know we are done with rules once we reach the first blank.

This allows us to feed our grammar to a Turing machine. In general we use the $\langle\ \rangle$ notation to represent the string encoding of some object. So for this case, we want to give $\langle G \rangle$ to our Turing machine M.

Our decider M will then work as follows:

```
$M$ = "On input $\langle G \rangle$:

    1. Skip to the rules and mark each terminal found. (We know what are terminals from
       the start of the tape.)
    2. Until no new variables are marked, for each rule do the following:
        (a) If every symbol from the RHS of our rule is marked, mark the LHS.
        (b) Scan all the rules and mark every occurrence of the LHS.
    3. If the start variable is marked, *accept*. Otherwise, *reject*."
```

For example, after the first step our tape looks like:

S	A	B	$\#$	a	b	ϵ	$\#$	S	A	$\#$	S	B	$\#$	S	A	B	$\#$	

(continued on next line...)

A	$\dot{a}$	A	$\#$	A	$\dot{\epsilon}$	$\#$	B	$\dot{b}$	B	$\#$	B	$\dot{\epsilon}$	$\#$	$\llcorner\!\lrcorner$	$\cdots$

Again, the devil is in the details. For example, how do we know if no new variables have been marked? We can set aside another cell on our tape. Before we start our scan, write a 0 in this field. If we mark a symbol, write a 1 in this cell. If there is still a 0 in the cell, then no new symbols have been marked.

Closure Properties

Show that the collection of decidable languages is closed under union.

Sipser Problem 3.15 (a)

We want to show that if L_1 and L_2 are decidable, then $L_1 \cup L_2 = L_3$ is decidable.

Let L_1 and L_2 be decidable languages, and M_1 and M_2 be the Turing machines that decide them. Let build M' as follows:

```
$M'$ = "On input $w$:
    1. Run $M_1$ on $w$. If it accepts, output *accept*.
    2. Run $M_2$ on $w$. If it accepts, output *accept*.
    3. Otherwise, output *reject*."
```

Now, we want to argue that $L(M') = L_1 \cup L_2$. To do this we must show that $L(M')$ accepts *only* those strings that are in $L_1 \cup L_2$. By the definition of union, we know that $w \in L_1 \cup L_2$ iff $w \in L_1$ or $w \in L_2$.

If $w \in L(M')$, then w is accepted by M_1 in step 1 or M_2 in step 2. Therefore $w \in L_1$ or $w \in L_2$, which by definition means $w \in L_1 \cup L_2$.

If $w \in L_1 \cup L_2$, then it is accepted by either M_1 or M_2. If it is accepted by M_1, it is accepted by M' in step 1. If it is accepted by M_2, it is accepted by M' in step 2. Therefore, $w \in L(M')$.

Show that the collection of Turing-recognizable languages is closed under union.

Sipser Problem 3.16 (a)

What is wrong with using the same proof? In this case, M_1 and M_2 are acceptors and may never halt. Consider the case where M_1 loops on w but M_2 accepts w. Then M' should accept w. However, if we use the same machine M', we will get stuck on step 1 and loop forever. This means M' will never accept w!

Instead we use a technique called dovetailing, is similar to the concept of multithreading. We run M_1 on the first symbol of w, and then switch to M_2. We run M_2 on the first symbol of w, and then switch back to M_1. We pick off where we started on M_1, and run M_1 on the second symbol of w. This continues to repeat until either M_1 or M_2 accept or reject.

The new machine M'' looks like:

```
$M''$ = "On input $w$:
    1. Run $M_1$ and $M_2$ alternatively on $w$ step by step (dovetail).
    2. If either $M_1$ or $M_2$ accept, output *accept*.
    3. If both $M_1$ and $M_2$ halt and reject, output *reject*."
```

If one of M_1 or M_2 loops, the other will still have a chance to run. Therefore, if either M_1 or M_2 accept, then M'' will accept. If they both halt and reject, then M'' will halt and reject. If they both loop, then M'' will also loop (making it an acceptor).

I'll leave a more complete argument that $L(M'') = L_1 \cup L_2$ for you to try out.

Decidability Proofs

Show that the following language is decidable:

$$\boldsymbol{INFINITE_{\text{DFA}}} = \{\, \langle A \rangle \mid A \text{ is a DFA and } L(A) \text{ is an infinite language} \,\}$$

Sipser Problem 4.9

To understand the algorithm for this one, there is some setup that is required.

First, notice that a DFA which accepts infinitely many strings must accept arbitrarily long strings. (If a DFA accepted only strings of length n or less, the language would be finite.)

Second, review the proof on page 79 of your book. Let p be the number of states in A. Consider a string s where $|s| = n \geq p$. The sequence of states A enters while processing s is $n + 1$ (since it enters the start state before processing a symbol). However, $n + 1 \geq p + 1$. By the pigeonhole principle, A must revist at least one state in the sequence. This indiates there is a loop, and tada... we can split s into three parts at satisfies the pumping lemma. (See the book for more detail.) As such, if A accepts s, then it accepts infinitely many strings since s may be pumped.

However, we have a problem. We can't test infinitely many strings s! How will we know when to stop testing strings?

Consider the language $L = \{ s \mid s \in \Sigma^* \text{ and } |s| \geq p \}$, which is the language of all strings of length p or more. This language is regular (it is essentially $\Sigma^p \, \Sigma^*$). Therefore we can create a DFA, B, where $L(B) = L$.

Now, what can we say about the intersection $L(A) \cap L(B)$? If it is empty, then A never accepts a string s where $|s| \geq p$. However, if it is nonempty, then A accepts such a string and is infinite.

Therefore, we create the TM M which decides $INFINITE_{\text{DFA}}$ as follows:

```
$M$ = "On input $\langle A \rangle$ where $A$ is a DFA:
    1. Let $p$ be the number of states in $A$.
    2. Construct a DFA $B$ that accepts all strings of length $p$ or more.
    3. Construct a DFA $C$ such that $L(C) = L(A) \cap L(B)$.
    4. Test if $L(C) = \emptyset$ (use $E_{\text{DFA}}$ from book).
    5. If $L(C)$ is empty, output *reject.*
    6. Otherwise, output *accept.*"
```