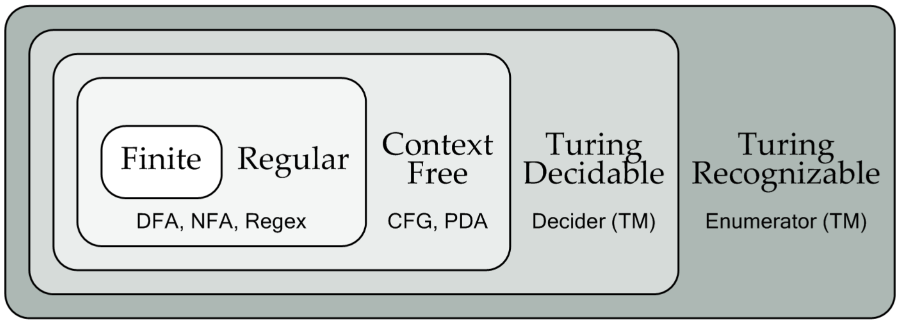
ECS120 Fall 2006

Discussion Notes

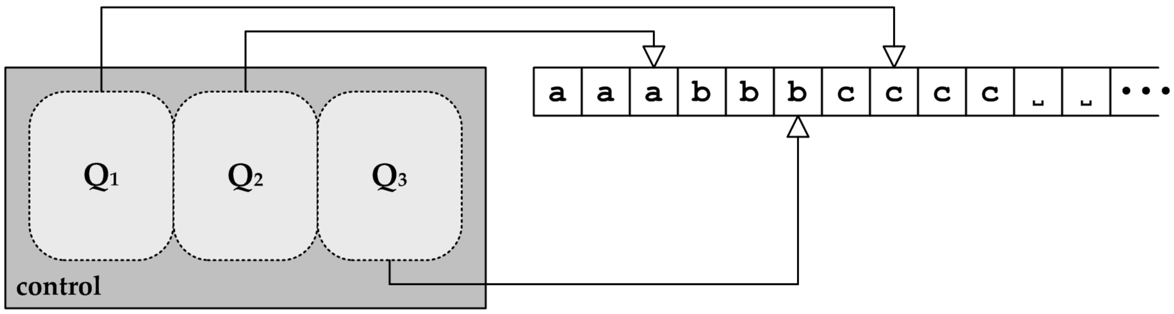
November 15, 2006

Language Review



Turing Machine Variants

Consider a **Multi-Head Turing Machine**. This is a single-tape Turing machine with multiple read/write heads. However, in every state only one head may be used. Therefore, for k heads we have a partition of states $Q_1, Q_2, \ldots, Q_k$ where each Q_i contains the set of states which use the ith head. For $k = 3$, we might have something like:



Example Usage

This variant is a useful for certain problems. Consider the language $L = \{a^n b^n c^n \mid n \geq 0\}$. We know this language is not context-free, but we can build a multi-head Turing machine with $k = 3$ to recognize this language. Lets call this machine M, which works as follows:

$M =$ "On input w:

1. Initialize each head as follows:
   (a) Let head 1 rest on the first input symbol of w.
   (b) Let head 2 scan the input for consecutive a symbols until the first b is encountered.
   (c) Let head 3 scan the input for consecutive a symbols, followed by consecutive b symbols, until a c is encountered.

   At this point head 1 should rest on the first a symbol, head 2 should rest on the first b symbol, and head 3 should rest on the first c symbol.
2. Until the end of string is encountered, loop as follows:
   (a) Test if head 1 rests on an a symbol and move right.
   (b) Test if head 2 rests on a b symbol and move right.
   (c) Test if head 3 rests on a c symbol and move right.

   If head 3 rests on a $\sqcup$ symbol, then the end of the string is has been encountered and we go on to the next step.
3. Accept if at the end of the string we have the following:
   (a) Head 1 rests on the first b symbol.
   (b) Head 2 rests on the first c symbol.
   (c) Head 3 rests on the first $\sqcup$ symbol.

   Otherwise, reject the string if any of the tests fail."

There are some missing details. For example, in the initialization phase if head 2 does not find consecutive a symbols at the start of the string, we reject. Or, if during the loop head 1 rests on a b symbol before the end of the string is reached, then we reject.

Equivalence

A multi-head Turing machine is equivalent to a normal Turing machine. Let M_{MH} be a multi-head Turing machine and M_{TM} be a normal Turing machine. To show that they are equivalent, we must provide two arguments.

Argument 1: For any normal Turing machine M_{TM}, there exists some multi-head Turing machine M_{MH} such that $L(M_{\text{TM}}) = L(M_{\text{MH}})$.

To show this, we need to show that M_{TM} can be simulated by some M_{MH}. This part of the proof is simple. We can build M_{MH} such that it forces all the states to use the same read/write head. The rest of the heads can be ignored. The states and transitions remain the same as in M_{TM}.

Argument 2: For any multi-head Turing machine M_{MH}, there exists some normal Turing machine M_{TM} such that $L(M_{\text{MH}}) = L(M_{\text{TM}})$.

We must show that M_{MH} can be simulated by some M_{TM}. This part is trickier. We could do something similar to the proof for a multi-tape Turing machine, and use some modifier on the symbols to track where the different heads are located on the tape. For example:

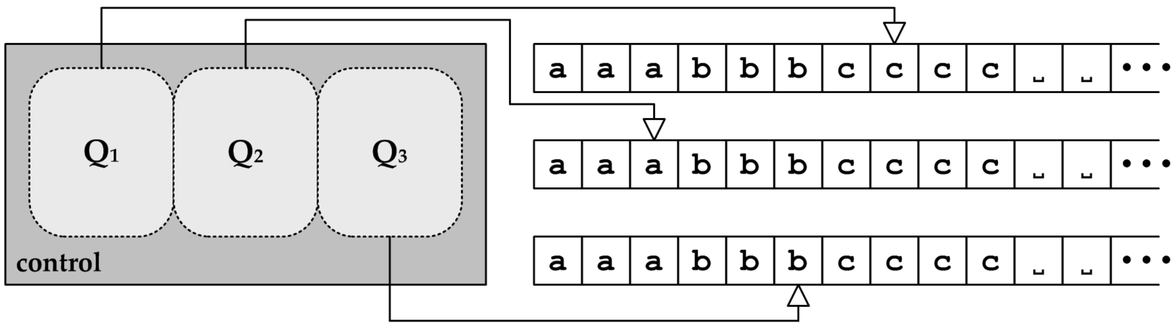
a	a	ȧ	b	b	ḅ	c	c	c	c'	␣	···

However, we already know multi-tape Turing machines are equivalent to normal Turing machines. We could alternatively show that for any multi-head Turing machine M_{MH}, there exists some multi-tape Turing machine M_{MT} such that $L(M_{\text{MH}}) = L(M_{\text{MT}})$.

Lets create a k-tape Turing machine M_{MT} to simulate a k-head Turing machine M_{MH} as follows:

1. Initialize every tape in M_{MT} with the input string.
2. Whenever head i is moved in M_{MH}, move the head in the ith tape in M_{MT}.
3. Whenever any head in M_{MH} writes a symbol, write the same symbol on the same position on every tape in M_{MT}.

Everything else would be the same in M_{MT} as in M_{MH}. So we would end up with something like:


Q1
Q2
Q3
control
a a a b b b c c c c ␣ ␣ •••
a a a b b b c c c c ␣ ␣ •••
a a a b b b c c c c ␣ ␣ •••

Since we know $L(M_{\text{MT}}) = L(M_{\text{TM}})$, we can conclude $L(M_{\text{MH}}) = L(M_{\text{MT}}) = L(M_{\text{TM}})$. Of course, all we have given here is an abstract idea of how to do this. The devil is in the details. You need to convince yourself that every step is correct.

Decision Procedure

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	$\dot{\epsilon}$	#	$\sqcup$	$\cdots$

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.