Lecture 11:
Turing Machines as Language Acceptors

Lecture Outline

- Defining the Languages TM's Accept
- Multiple Tape Turing Machines
- NonDeterministic Turing Machines
- The Equivalence of Turing Acceptable Languages and Phrase Structure Languages
- The Existence of Non-Phrase Structure Languages

Reading

Chapter 3.3 - 3.4 of Brookshear

Chapter 9 Martin, 2nd ed.

Chapter 6.4 of Revesz

Chapter 24 of Cohen

Chapter 7 of Hopcroft and Ullman

Defining the Languages TM's Accept

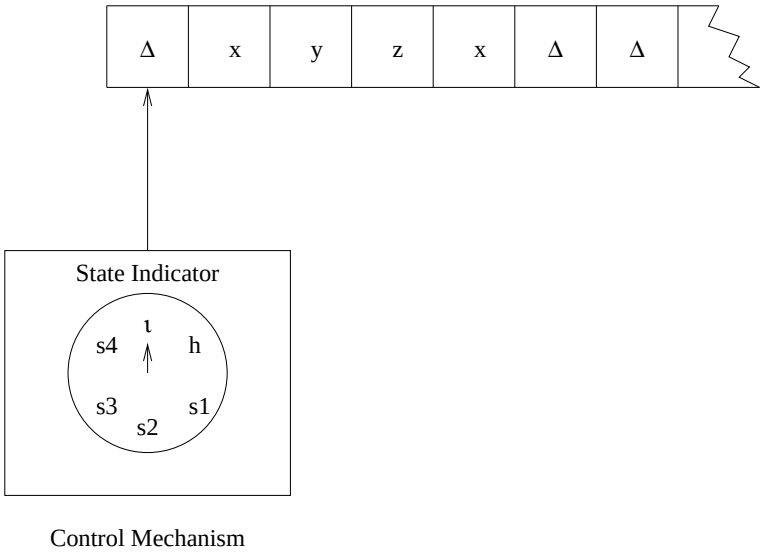
- To specify the languages a Turing Machine accepts we must specify how the string is to be presented to the machine.

- We suppose the string is recorded starting in the second cell from the left end of the machine's tape.

  The tape is assumed to be all blank with the exception of the string to be tested (recall that the blank is assumed not to be in the alphabet from which the strings to be tested are drawn).

- The machine's tape head is assumed to be initially positioned over the leftmost cell on the tape and the machine started in the initial state.

- The machine is said to accept the string if starting from this initial configuration it finds its way to a halt state.



Initial configuration for a TM testing the string $xyzx$.

Defining the Languages TM's Accept (cont)

- The language accepted by a Turing Machine M is denoted $L(M)$.

  A language L for which there is a Turing Machine M such that $L(M) = L$ is called a **Turing-acceptable language**.

- How does the class of Turing-acceptable languages relate to the other classes of languages (those accepted by finite automata, or pushdown automata) that we have studied ?

  We shall see that they accept a larger class than the other automata we have examined.

- It can be useful to insist that a TM write an acceptance message on its tape before halting.

  E.g. we might demand that a TM accept a string only by halting with the tape configuration

  $$\underline{\Delta} Y \Delta\Delta\Delta \cdots$$

  where the Y signifies the response "yes".

- Does the requirement to produce an acceptance message affect the class of strings recognised ? No.

  > *For every TM M which accepts strings simply by halting there is another TM M' which accepts strings by halting with tape configuration $\underline{\Delta} Y \Delta\Delta\Delta \cdots$ such that $L(M) = L(M')$.*

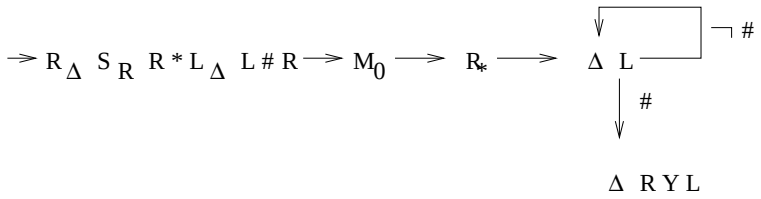
Defining the Languages TM's Accept (cont)

- Given a machine M which accepts strings simply by halting, a machine M' which accepts the same language but writes an accept message before halting can be specified in the following way.
- Suppose we start with the configuration $\underline{\Delta}w\Delta\Delta\cdots$ where w is the string to be recognised.

  Then M'

  1. starts by shifting the string one position to the right and writing a # before the blank prior its first symbol and a * after its last symbol (assume # and * do not occur in the input alphabet)
  2. simulates the behaviour of M precisely except when
     (a) the # is encountered – as this would mean an abnormal termination for M, M' should abnormally terminate here by moving the tape head one position further to the left;
     (b) the * is encountered – this means the tape head is moving onto previously unused tape; in this case the * should be moved along one position further to the right.
  3. when the simulation of M reaches M's halt state, finishes by erasing that portion tape of the tape between the # and *, writing a Y in position 2, then halting.

  More precisely M' would be as follows, where M_0 is the just like M save for the two conditions 2 (a) and (b).



Multiple-Tape Turing Machines

- Before considering other ways of characterising the languages accepted by Turing machines, it is worth considering ways in which the power of Turing Machines might be extended.

- One way in which we might suppose the power of a TM could be extended is to give the machine multiple tapes to write upon.

- A **k-tape Turing machine**, $k > 1$, is just like a 1-tape TM except:

  1. the machine has k tapes each of which
     - has a fixed left end and stretches indefinitely to the right
     - has a read-write head;
  2. transitions are dependent on the collection of current symbols on each of the tapes (i.e. those under the read-write heads) together with the current state of the machine;
  3. the action resulting from a transition affects exactly one of the tapes – either a symbol is written on it, or its tape head moves left or right.

- To test a string with a k-tape machine, the string is written on the machine's first tape (as with a 1-tape machine), the other tapes are all blank, and all tape heads are positioned in the leftmost cell of their tapes.

  If the machine halts when started in the initial state with this configuration then the string is accepted.

- Surprisingly, this result holds:

  Proposition 11.1 . *For each multiple-tape Turing machine M there is a single tape Turing machine M' such that $L(M) = L(M')$.*

I.e. the class of languages accepted by TM's is **not** affected by the number of tapes.

Nondeterministic Turing Machines

- Another way in which it might be thought the power of a TM could be extended is to make the machine nondeterministic.

  Recall that:

  - nondeterministic finite automata **are not** more powerful than their deterministic counterparts,
  - nondeterministic pushdown automata **are** more powerful than deterministic pushdown automata.

- A **nondeterministic** Turing machine is just like a deterministic TM except that there may be more than one transition for a given current state/symbol pair.

  Thus, instead of a transition *function* δ there is a transition *relation* ρ.

  It follows that the class of deterministic TM's is a subset of the class of nondeterministic TM's.

- A string is accepted by a nondeterministic TM if it is possible for the machine to reach a halt state starting with the string in the standard testing configuration.

  That is, there must be *some* sequence of allowable transitions that allows the machine to reach a halt state (though some other sequences might not).

- Nondeterminism, however, does not increase the power of a TM:

  Proposition 11.2 *For each nondeterministic Turing machine N there is a deterministic Turing machine D such that $L(N) = L(D)$.*

Turing-Acceptable Languages and Phrase-Structure Languages

- Recall our definition of phrase-structure grammars as four-tuples consisting of:

  1. a finite set of nonterminal symbols;
  2. a finite set of terminal symbols (disjoint from the nonterminals);
  3. a start symbol (one of the nonterminals);
  4. a set of rewrite rules of the form $A \rightarrow P$ where A and P are strings of nonterminals and terminals and A contains at least one nonterminal.

- Different sorts of grammars can be specified by stipulating constraints on the form of the rewrite rules:

  1. **Regular** grammars impose the condition that in the rewrite rules A had to be a single nonterminal and P had to be either a single terminal followed by a single nonterminal, a single terminal, or the empty string.
  2. **Context-free** grammars impose the condition that in the rewrite rules A had to be a single nonterminal and P was any string of terminals and nonterminals.

  In their most general form, however, the only constraint imposed by **phrase-structure grammars** is that the lefthand side contain at least one nonterminal.

- Any language that can be generated using a phrase-structure grammar is called a **phrase-structure language**, reflecting the fact that it can be specified in terms of a hierarchy of phrase structures.

Turing-Acceptable Languages and Phrase-Structure Languages (cont)

- We know there are languages that are not context-free:

$$\{x^n y^n z^n \mid n \in \mathbf{N}\}.$$

- This language is generated by a phrase structure grammar:

$$\begin{aligned} S &\rightarrow xyNSz \\ S &\rightarrow \lambda \\ yNx &\rightarrow xyN \\ yNz &\rightarrow yz \\ yNy &\rightarrow yyN \end{aligned}$$

  Thus the class of phrase structure languages is strictly larger than the class of context-free languages.

- Our final result about machine-class/language-class equivalence is to show that *the phrase structure languages are exactly equivalent to the Turing acceptable languages*.

- To establish this claim requires proving two propositions:

  Proposition 11.3 *Every Turing-acceptable language is a phrase-structure language.*

  Proposition 11.4 *Every phrase-structure language is a Turing-acceptable language.*

The Equivalence of Turing-Acceptable and Phrase-Structure Languages

- As a preliminary we define a notation for representing the entire configuration of a TM at any point. We write

$$[x_1 \cdots x_i \; p \; x_{i+1} \cdots x_n \Delta]$$

where

1. [indicates the left end of the machine's tape
2. $x_1 \cdots x_i$ is a string (possibly empty) of tape symbols located in cells 1 to i;
3. p denotes the current state of the machine;
4. $i + 1$ is the cell over which the tape head is currently positioned;
5. $x_{i+1} \cdots x_n$ is a string (possibly empty) of tape symbols located in cells $i + 1$ to n;
6.] indicates that all cells further to the right are blank.

For example:

$$[\Delta xyypzy\Delta]$$

indicates the machine's tape holds the symbols $\Delta xyyzy\Delta\Delta\cdots$, that the current state is p and that the tape head is positioned over the first z.

- If a machine accepts a string by writing an acceptance message its behaviour can be represented as a sequence of configurations of this form starting, for a string w with

$$[\iota\Delta w\Delta]$$

and finishing with

$$[h\Delta Y\Delta]$$

- Such a sequence, read backwards, corresponds to a grammatical derivation ...

The Equivalence of Turing-Acceptable and Phrase-Structure Languages (cont)

Proposition 11.3 *Every Turing-acceptable language is a phrase-structure language.*

Proof

Let L be a Turing-acceptable language and let M be a TM such that $L(M) = L$ and such that M accepts strings by halting with its tape configured as $\underline{\Delta}Y\Delta\Delta$.

Define a grammar G as follows:

1. The start symbol of G is S.
2. The nonterminals of G are
   (a) S, [,];
   (b) symbols representing the states of M;
   (c) those tape symbols of M, including Δ and Y, *not* in the input alphabet of M.
3. The terminals of G are the symbols in the input alphabet of M.
4. The rewrite rules of G are:
   (a) $S \rightarrow [h\Delta Y\Delta]$
   So the derivation will start with the halt configuration.
   (b) $\Delta] \rightarrow \Delta\Delta]$
   This allows the halt configuration to be expanded to any length.
   (c) for each transition of the form $\delta(p, x) = (q, y)$ a rewrite rule $qy \rightarrow px$
   E.g. $[\Delta zqy\Delta]$ could be rewritten as $[\Delta zpx\Delta]$ corresponding to the transition of M from $[\Delta zpx\Delta]$ to $[\Delta zqy\Delta]$ by the transition $\delta(p, x) = (q, y)$.

The Equivalence of Turing-Acceptable and Phrase-Structure Languages (cont)

(d) for each transition of the form $\delta(p, x) = (q, R)$ a rewrite rule $xq \rightarrow px$
E.g. $[\Delta xqyz\Delta]$ could be rewritten as $[\Delta pxyz\Delta]$.

(e) for each transition of the form $\delta(p, x) = (q, L)$ and each tape symbol y of M a rewrite rule $qyx \rightarrow ypx$
E.g. $[\Delta qyx\Delta]$ could be rewritten as $[\Delta ypx\Delta]$.

(f) $\iota\Delta \rightarrow \lambda$, $\Delta\Delta] \rightarrow \Delta]$, $\Delta] \rightarrow \lambda$
So if the derivation produces the initial configuration $[\iota\Delta xyz\Delta\Delta]$ then the nonterminals could be removed to yield the string xyz.

Now we show $L(M) = L(G)$. For, suppose w is a string in $L(M)$. Then there is a sequence of configurations of M starting with $[\iota\Delta w\Delta]$ and ending with $[h\Delta Y\Delta]$.

So, there is a derivation of w of the form:

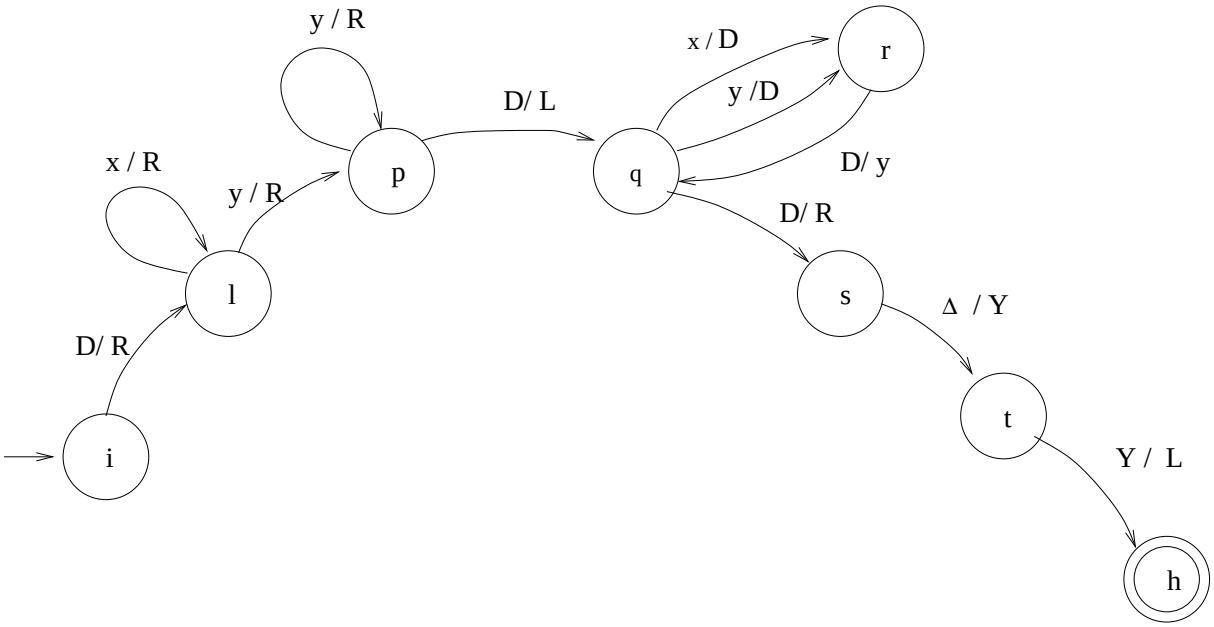
$$S \Rightarrow [h\Delta Y\Delta] \Rightarrow \cdots \Rightarrow [\iota\Delta w\Delta] \Rightarrow w\Delta] \Rightarrow w$$

First apply the rule $S \rightarrow [h\Delta Y\Delta]$, then apply the rule $\Delta] \rightarrow \Delta\Delta]$ as many times as necessary to grow the string $[h\Delta Y\Delta \cdots \Delta]$ as long as any configuration in the computation of M.

Next apply the transitions in the original computation sequence in reverse order until the pattern $[\iota\Delta w\Delta \cdots \Delta]$ is obtained. Then reduce this to w using the rules in 4 (f). It follows that w is in $L(G)$.

Conversely, suppose $w \in L(G)$. Then there is a derivation of w in $L(G)$ and this derivation can be used to construct a sequence of configurations which would demonstrate how M could accept w. Thus, any w $L(G)$ is also in $L(M)$. ∎

Example (Brookshear)



Grammar

$$
\begin{array}{lcl}
S & \rightarrow & [h\Delta Y\Delta] \\
\Delta] & \rightarrow & \Delta\Delta] \\
[\iota\Delta & \rightarrow & \lambda \\
\Delta\Delta] & \rightarrow & \Delta] \\
\Delta] & \rightarrow & \lambda \\
h\Delta Y & \rightarrow & \Delta tY \\
hxY & \rightarrow & xtY \\
hyY & \rightarrow & ytY \\
hYY & \rightarrow & YtY \\
tY & \rightarrow & s\Delta \\
qy & \rightarrow & r\Delta \\
r\Delta & \rightarrow & qx \\
r\Delta & \rightarrow & qy \\
qy\Delta & \rightarrow & yp\Delta \\
qx\Delta & \rightarrow & xp\Delta \\
q\Delta\Delta & \rightarrow & \Delta p\Delta \\
qy\Delta & \rightarrow & Yp\Delta \\
yp & \rightarrow & py \\
yp & \rightarrow & ly \\
xl & \rightarrow & lx \\
\Delta l & \rightarrow & \iota\Delta
\end{array}
$$

Machine Configuration Sequence

$$
\begin{array}{l}
[\iota\Delta xxy\Delta] \\
[\Delta lxxy\Delta] \\
[\Delta xlxy\Delta] \\
[\Delta xxly\Delta] \\
[\Delta xxyp\Delta] \\
[\Delta xxqy\Delta] \\
[\Delta xxr\Delta] \\
[\Delta xqx\Delta] \\
[\Delta xr\Delta] \\
[\Delta qx\Delta] \\
[\Delta r\Delta] \\
[q\Delta] \\
[\Delta s\Delta] \\
[\Delta tY\Delta] \\
[h\Delta Y\Delta]
\end{array}
$$

Grammatical Derivation

$$
\begin{array}{lcl}
S & \Rightarrow & [h\Delta Y\Delta] \\
 & \Rightarrow & [h\Delta Y\Delta\Delta] \\
 & \Rightarrow & [h\Delta Y\Delta\Delta\Delta] \\
 & \Rightarrow & [\Delta tY\Delta\Delta\Delta] \\
 & \Rightarrow & [\Delta s\Delta\Delta\Delta\Delta] \\
 & \Rightarrow & [q\Delta\Delta\Delta\Delta\Delta] \\
 & \Rightarrow & [\Delta r\Delta\Delta\Delta\Delta] \\
 & \Rightarrow & [\Delta qx\Delta\Delta\Delta] \\
 & \Rightarrow & [\Delta xr\Delta\Delta\Delta] \\
 & \Rightarrow & [\Delta xqx\Delta\Delta] \\
 & \Rightarrow & [\Delta xxr\Delta\Delta] \\
 & \Rightarrow & [\Delta xxqy\Delta] \\
 & \Rightarrow & [\Delta xxyp\Delta] \\
 & \Rightarrow & [\Delta xxly\Delta] \\
 & \Rightarrow & [\Delta xlxy\Delta] \\
 & \Rightarrow & [\Delta lxxy\Delta] \\
 & \Rightarrow & [\iota\Delta xxy\Delta] \\
 & \Rightarrow & xxy\Delta] \\
 & \Rightarrow & xxy
\end{array}
$$

The Equivalence of Turing-Acceptable Languages and Phrase-Structure Languages (cont)

Proposition 11.4 *Every phrase-structure language is a Turing-acceptable language.*

Proof

The proof relies on the propositions:

1. for every nondeterministic TM N there is a deterministic TM D such that $L(N) = L(D)$ (Proposition 11.2);
2. for every multiple tape nondeterministic TM M there is a one tape nondeterministic TM M' such that $L(M) = L(M')$ (follows from a slight modification of the proof of Proposition 11.1).

We argue that for each grammar G there is a nondeterministic two-tape TM N such that $L(G) = L(M)$. It then follows by the above observations that there is a deterministic one tape machine D such that $L(G) = L(D)$.

Note that any rewrite rule $A \rightarrow P$ in the grammar can be implemented by a TM: if the terminals and nonterminals making up A occur on the machine's tape then by using the basic operations of left and right shifting and writing the machine can replace A with P.

The Equivalence of Turing-Acceptable Languages and Phrase-Structure Languages (cont)

Construct a nondeterministic two-tape TM N that works as follows:

1. write the string to be tested on tape 1
2. write the grammar's start symbol on tape 2
3. apply the rewrite rules of the grammar to the string on tape 2 in a *nondeterministic* fashion – recall that there could be several applicable rules for any nonterminal
4. if a string consisting solely of terminals appears on tape 2 compare the string to the string on tape 1: if they are the same halt; otherwise move a tape head to the left until abnormal termination occurs.

Note: this algorithm is a form of *generate and test* algorithm.

Any string derivable from G may be produced on tape 2. Hence if a string w in $L(G)$ is placed on tape 1 then it is possible for N to halt (the definition of string acceptance for nondeterministic TM's).

If a string w not derivable from G is placed on tape 1 then since *only* strings derivable from G may occur on tape 2 then the string on tape 2 can never match w and N cannot accept the string.

It follows that $L(G) = L(N)$.

∎

The Existence of Non-Phrase Structure Languages

- Suppose L is a phrase structure language over alphabet Σ and M is a TM such that $L(M) = M$. From M we can always derive a TM M' with tape symbols $\Sigma \cup \{\Delta\}$ such that $L(M) = L(M')$.

  That is, Δ is the only tape symbol needed aside from the symbols of the alphabet Σ.

- To see this argue as follows. Let x be a symbol from Σ.

  By arranging all nonblank tape symbols of M in a list and assigning x to the first list entry, xx to the second, and so on, any tape symbol of M can be represented by a unique string on x's.

  Any string of symbols of which occurs on M's tape can be represented by these new strings of x's separated by blanks, where genuine blanks are now represented by two consecutive blanks.

  M' is constructed so that it translates its input into this coded form, then simulates M, halting if only if M halts.

  Thus M' accepts exactly the strings that M does, but uses only the tape symbols $\Sigma \cup \{\Delta\}$.

- All Turing machines with tape symbols $\Sigma \cup \Delta$ can be systematically listed by listing first those with two states (minimum number for TMs), then those with three states, etc.

- However the number of strings in Σ^* is infinite and the number of languages over Σ is uncountably infinite (the power set theorem).

- Therefore there are more languages over Σ than there are Turing machines with tape symbols $\Sigma \cup \{\Delta\}$.

- But since for every *phrase structure* language over Σ there is a TM with tape symbols $\Sigma \cup \{\Delta\}$ which accepts it, it follows that there are languages over Σ which are not phrase structure languages.