The Chomsky Hierarchy and Beyond

Chapter 24
Is There Anything In Between PDAs and Turing Machines?

PDAs aren’t powerful enough.

Turing machines lack even a decision procedure for the acceptance problem.
A linear bounded automaton is an NDTM the length of whose tape is equal to \(|w| + 2\).

Example: \(A^nB^nC^n = \{a^nb^n c^n : n \geq 0\}\)
Linear Bounded Automata

A language is \textit{context sensitive} iff there exists an LBA that accepts it.

Note: It is not known whether, for every nondeterministic LBA there exists an equivalent deterministic one.
The Membership Question for LBAs

Let $L = \{<B, w> : \text{LBA } B \text{ accepts } w\}$. Is $L$ in $D$?

How many distinct configurations of $B$ exist?
The Membership Question for LBAs

Let \( L = \{<B, w> : \text{LBA } B \text{ accepts } w\} \). Is \( L \) in \( D \)?

\[
\begin{array}{cccccc}
... & \blacklozenge & a & b & b & a & \blacklozenge & ... \\
\end{array}
\]

\( q_0 \)

How many distinct configurations of \( B \) exist?

\[
\text{MaxConfigs} = |K| \cdot |\Gamma|^{(|w|+2)} \cdot (|w| + 2)
\]
The Membership Question for LBAs

**Theorem:** \( L = \{<B, w> : \text{LBA } B \text{ accepts } w\} \) is in \( D \).

**Proof:** If \( B \) runs for more than \( \text{MaxConfig} \) steps, it is in a loop and it is not going to halt.

\( M \) is an NDTM that decides \( L \):

\[
M(<B, w>) =
\]

1. Simulate all paths of \( B \) on \( w \) for \( \text{MaxConfig} \) steps or until \( B \) halts, whichever comes first.
2. If any path accepted, accept. Else reject.

Since, from each configuration of \( B \) there are a finite number of branches and each branch is of finite length, \( M \) will be able to try all branches of \( B \) in a finite number of steps. \( M \) will accept the string \( <B, w> \) if any path of \( B \) accepts and it will reject the string \( <B, w> \) if every path of \( B \) on \( w \) either rejects or loops.
Grammars, Context-Sensitive Languages, and LBAs

Grammar $L$ accepts CS Language

CS Language

LBA

Accepts
Context-Sensitive Grammars and Languages

A **context-sensitive grammar** \( G = (V, \Sigma, R, S) \) is an unrestricted grammar in which \( R \) satisfies the following constraints:

- The left-hand side of every rule contains at least one nonterminal symbol.

- No length-reducing rules, with one exception:

  Consider:

  \[
  \begin{align*}
  S \rightarrow & \ a \ S \ b \\
  S \rightarrow & \ \varepsilon \quad \text{/* length reducing}
  \end{align*}
  \]
Context-Sensitive Grammars and Languages

A context-sensitive grammar $G = (V, \Sigma, R, S)$ is an unrestricted grammar in which $R$ satisfies the following constraints:

- The left-hand side of every rule contains at least one nonterminal symbol.

- No length-reducing rules, with one exception:
  - $R$ may contain the rule $S \rightarrow \varepsilon$.

  If it does, then $S$ does not occur on the right hand side of any rule.
Context-Sensitive Grammars and Languages

Example of a grammar that is not context-sensitive:

\[
S \rightarrow aSb \\
S \rightarrow \varepsilon
\]

An equivalent, context-sensitive grammar:
Context-Sensitive Grammars and Languages

Example of a grammar that is not context-sensitive:

\[
\begin{align*}
S & \rightarrow aSb \\
S & \rightarrow \varepsilon
\end{align*}
\]

An equivalent, context-sensitive grammar:

\[
\begin{align*}
S & \rightarrow \varepsilon \\
S & \rightarrow T \\
T & \rightarrow ab \\
T & \rightarrow aTb
\end{align*}
\]
A^nB^nC^n

A^nB^nC^n

S \rightarrow aBS c
S \rightarrow \varepsilon
B a \rightarrow a B
Bc \rightarrow bc
B b \rightarrow bb

/* Not a CS rule
Equal Numbers of $a$’s, $b$’s, and $c$’s

\[
\{w \in \{a, b, c\}^*: \#_a(w) = \#_b(w) = \#_c(w)\}
\]

\[
\begin{align*}
S & \rightarrow ABCS \\
S & \rightarrow \varepsilon \\
AB & \rightarrow BA \\
BC & \rightarrow CB \\
AC & \rightarrow CA \\
BA & \rightarrow AB \\
CA & \rightarrow AC \\
CB & \rightarrow BC \\
A & \rightarrow a \\
B & \rightarrow b \\
C & \rightarrow c
\end{align*}
\]
WW

WW = \{ww : w \in \{a, b\}^*\}

\begin{align*}
S & \rightarrow T# \\
T & \rightarrow aTa \\
T & \rightarrow bTb \\
T & \rightarrow C \\
C & \rightarrow CP \\
P_{aa} & \rightarrow aPa \\
P_{ab} & \rightarrow bPa \\
P_{ba} & \rightarrow aPb \\
P_{bb} & \rightarrow bPb \\
P_{a#} & \rightarrow #a \\
P_{b#} & \rightarrow #b \\
C# & \rightarrow \varepsilon \\
\end{align*}

/* Generate the wall exactly once. 
/* Generate \(wCwR\). 
   " 
   " 

/* Generate a pusher \(P\) 
/* Push one character to the right to get ready to jump. 
   " 
   " 

/* Hop a character over the wall. 
   "
The Membership Question for Context-Sensitive Grammars

Let \( L = \{<G, w>: \text{csg } G \text{ generates string } w\} \). Is \( L \) in \( D \)?

Example:

\[ S \rightarrow aBSc \]
\[ S \rightarrow aBc \]
\[ Ba \rightarrow aB \]
\[ Bc \rightarrow bc \]
\[ Bb \rightarrow bb \]

\( S \Rightarrow aBSc \Rightarrow aBaBScc \Rightarrow aaBBScc \ldots \)
\[ L = \{<G, w> : \text{CSG } G \text{ generates string } w\} \text{ is in } D. \]

**Proof:** We construct an NDTM \( M \) to decide \( D \). \( M \) will explore all derivations that \( G \) can produce starting with \( S \). Eventually one of the following things must happen on every derivation path:

- \( G \) will generate \( w \).
- \( G \) will generate a string to which no rules can be applied.
- \( G \) will keep generating strings of the same length. Since there are a finite number of strings of a given length, \( G \) must eventually generate the same one twice. The path can be terminated since it is not getting any closer to generating \( w \).
- \( G \) will generate a string \( s \) that is longer than \( w \).

Since \( G \) has only a finite number of choices at each derivation step and since each path that is generated must eventually halt, the Turing machine \( M \) that explores all derivation paths will eventually halt. If at least one path generates \( w \), \( M \) will accept. If no path generates \( w \), \( M \) will reject.
Context-Sensitive Languages and Linear Bounded Automata

Theorem: The set of languages that can be generated by a context-sensitive grammar is identical to the class that can be accepted by an LBA.

Proof: (sketch)
- Given a CSG $G$, build a two-track LBA $B$ such that $L(B) = L(G)$. On input $w$, $B$ keeps $w$ on the first track. On the second track, it nondeterministically constructs all derivations of $G$. As soon as any derivation becomes longer than $|w|$, stop.
- From any LBA $B$, construct a CSG that simulates $B$. 
Languages and Machines

- SD
- D
- Context-sensitive
- Context-free
- DCF
- Regular FSMs
- DPDAs
- NDPDAs
- LBAs
- Turing machines
**Theorem:** The CS languages are a proper subset of D.

**Proof:** We divide the proof into two parts:

- Every CS language is in D: Every CS language \( L \) is accepted by some LBA \( B \). The Turing machine that performs a bounded simulation of \( B \) decides \( L \).

- There exists at least one language that is in D but that is not context-sensitive: It is not easy to do this by actually exhibiting such a language. But we can use diagonalization to show that one exists.
Using Diagonalization

Create an encoding for context-sensitive grammars:

\[ x00 \rightarrow ax00a; x00 \rightarrow x01; x01 \rightarrow b01b; x01 \rightarrow b \]

\(\text{Enum}_G\) is the lexicographic enumeration of all encodings of CSGs with \(\Sigma = \{a, b\}\).

\(\text{Enum}_{a,b}\) is the lexicographic enumeration of \(\{a, b\}^*\).

<table>
<thead>
<tr>
<th>Grammar</th>
<th>String 1</th>
<th>String 2</th>
<th>String 3</th>
<th>String 4</th>
<th>String 5</th>
<th>.......</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.....</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.....</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.....</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>.....</td>
</tr>
<tr>
<td>...</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>.....</td>
</tr>
</tbody>
</table>

Because \(\{<G, w> : \text{CSG } G \text{ generates string } w\}\) is in \(D\), there exists a TM that can compute the values in this table as they are needed.
Diagonalization, Continued

Define $L_D = \{string_i : string_i \notin L(G_i)\}$. $L_D$ is:

- Recursive because it is decided by the following Turing machine $M$:

  $M(x) =$
  
  1. Find $x$ in the list $Enum_{a,b}$. Let its index be $i$.
  2. Lookup cell $(i, i)$ in the table.
  3. If the value is 0, $x$ is not in $L(G_i)$ so $x$ is in $L$. Accept.
  4. If the value is 1, $x$ is in $L(G_i)$ so $x$ is not in $L$. Reject.

- Not context-sensitive because it differs, in the case of at least one string, from every language in the table and so is not generated by any context-sensitive grammar.
Theorem: The context-free languages are a proper subset of the context-sensitive languages.

Proof: We divide the proof into two parts:

- We know one language, $A^nB^nC^n$, that is context-sensitive but not context-free.

- If $L$ is a context-free language then there exists some context-free grammar $G = (V, \Sigma, R, S)$ that generates it:
  - Convert $G$ to $G'$ in Chomsky normal form
  - $G'$ generates $L - \{\epsilon\}$.
  - $G'$ has no length-reducing rules, so is a CSG.
  - If $\epsilon \in L$, add to $G'$ the rules $S' \rightarrow \epsilon$ and $S' \rightarrow S$.
  - $G'$ is still a context-sensitive grammar and it generates $L$.
  - So $L$ is a context-sensitive language.
Closure Properties

The context-sensitive languages are closed under:

- Union
- Concatenation
- Kleene star
- Intersection
- Complement
**Theorem:** The CSLs are closed under union.

**Proof:** By construction of a CSG \( G \) such that \( L(G) = L(G_1) \cup L(G_2) \):

If \( L_1 \) and \( L_2 \) are CSLs, then there exist CSGs:

\[
G_1 = (V_1, \Sigma_1, R_1, S_1) \quad \text{and} \quad G_2 = (V_2, \Sigma_2, R_2, S_2)
\]

such that \( L_1 = L(G_1) \) and \( L_2 = L(G_2) \).

Rename the nonterminals of \( G_1 \) and \( G_2 \) so that they are disjoint and neither includes the symbol \( S \).

\( G \) will contain all the rules of both \( G_1 \) and \( G_2 \). Add to \( G \) a new start symbol, \( S \), and two new rules:

\[
S \rightarrow S_1 \quad \text{and} \quad S \rightarrow S_2.
\]

So \( G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2,
R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S) \).
Closure Under Concatenation

**Theorem:** The CSLs are closed under concatenation.

**Proof:** By construction of a CSG $G$ such that $L(G) = L(G_1) \cdot L(G_2)$:

If $L_1$ and $L_2$ are CSLs, then there exist CSGs $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ such that $L_1 = L(G_1)$ and $L_2 = L(G_2)$.

Let $G$ contain all the rules of $G_1$ and $G_2$. Then add a new start symbol, $S$, and one new rule, $S \rightarrow S_1 S_2$.

Problem:

```
  S
 / \
S_1  S_2
   / \   /
  a a A a a
```

The subtrees may interact.
Nonterminal Normal Form

A CSG $G = (V, \Sigma, R, S)$ is in *nonterminal normal form* iff all rules in $R$ are of one of the following two forms:

- $\alpha \rightarrow c$, where $\alpha$ is an element of $(V - \Sigma)$ and $c \in \Sigma$,
- $\alpha \rightarrow \beta$, where both $\alpha$ and $\beta$ are elements of $(V - \Sigma)^+$.

$A \rightarrow aBBc$ is not in nonterminal normal form.
$aAB \rightarrow BB$ is not in nonterminal normal form.
Nonterminal Normal Form

**Theorem:** Given a CSG $G$, there exists an equivalent nonterminal normal form grammar $G'$ such that $L(G') = L(G)$.

**Proof:** The proof is by construction.

$converttononterminal(G: CSG) =$

1. Initially, let $G' = G$.
2. For each terminal symbol $c$ in $\Sigma$, create a new nonterminal symbol $T_c$. Add to $R_{G'}$ the rule $T_c \rightarrow c$.
3. Modify each of the original rules so that every occurrence of a terminal symbol $c$ is replaced by the nonterminal symbol $T_c$.

$AA \rightarrow aBBc$ becomes $AA \rightarrow T_aBBT_c$

$T_a \rightarrow a$

$T_c \rightarrow c$

$aAB \rightarrow BB$ becomes $T_aAB \rightarrow BB$
Closure Under Concatenation

Now the subtrees will be:

We prevent them from interacting by standardizing apart the nonterminals.
To build a grammar $G$ such that $L(G) = L(G_1) \cdot L(G_2)$, we do the following:

1. Convert both $G_1$ and $G_2$ to nonterminal normal form.
2. If necessary, rename the nonterminals of $G_1$ and $G_2$ so that the two sets are disjoint and so that neither includes the symbol $S$.
3. Let $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}, S)$. 

Concatenation, Continued
Decision Procedures

The membership problem is decidable, although no efficient procedure is known.

Other questions are not:

- Is $L$ empty?
- Is the intersection of two CSLs empty?
- Is $L = \Sigma^*$?
- Are two CSLs equal?
The Chomsky Hierarchy

- SD (Type 0)
- Context-Sensitive (Type 1)
- Context-Free (Type 2)
- Regular (Type 3)
- FSMs
- PDAs
- LBAs
- Turing Machines
The idea: constrain rule firing by:

- defining features that can be passed up/down in parse trees, and

- describing feature-value constraints that must be satisfied before the rules can be applied.
$A^nB^nC^n$

$G$ will exploit one feature, $size$.

$G = (\{S, A, B, C, a, b, c\}, \{a, b, c\}, R, S)$, where:

$$R = \{ S \rightarrow A B C \quad (size(A) = size(B) = size(C))$$

$$A \rightarrow a \quad (size(A) \leftarrow 1)$$

$$A \rightarrow A_2 a \quad (size(A) \leftarrow size(A_2) + 1)$$

$$B \rightarrow b \quad (size(B) \leftarrow 1)$$

$$B \rightarrow B_2 b \quad (size(B) \leftarrow size(B_2) + 1)$$

$$C \rightarrow c \quad (size(C) \leftarrow 1)$$

$$C \rightarrow C_2 c \quad (size(C) \leftarrow size(C_2) + 1) \}.$$

Applying $G$ bottom up:

```
   aaabbbcccc
```
A Unification Grammar for Subject/Verb Agreement

* The bear like chocolate.

The culprit is the rule $S \rightarrow NP \ VP$. Replace $NP$ and $VP$ by:

\[
[ \text{CATEGORY NP} \quad \text{CATEGORY VP} ]
\begin{array}{ll}
\text{PERSON} & \text{THIRD} \\
\text{NUMBER} & \text{SINGULAR}
\end{array}
\begin{array}{ll}
\text{PERSON} & \text{THIRD} \\
\text{NUMBER} & \text{SINGULAR}
\end{array}
\]

Replace atomic terminal symbols like bear, with:

\[
[ \text{CATEGORY N} ]
\begin{array}{ll}
\text{LEX} & \text{bear} \\
\text{PERSON} & \text{THIRD} \\
\text{NUMBER} & \text{SINGULAR}
\end{array}
\]

A Unification Grammar for Subject/Verb Agreement

Replace $S \rightarrow NP \ VP$ with:

$$
[\text{CATEGORY } S] \rightarrow
[\text{CATEGORY } NP \quad [\text{CATEGORY } VP
\text{NUMBER} \quad x_1 \\
\text{PERSON} \quad x_2 ] \quad \text{NUMBER} \quad x_1 \\
\text{PERSON} \quad x_2 ]$$

So an $NP$ and a $VP$ can be combined to form an $S$ iff they have matching values for their $\text{NUMBER}$ and $\text{PERSON}$ features.
Lindenmayer Systems

An L-system $G$ is a triple $(\Sigma, R, \omega)$, where:

- $\Sigma$ is an alphabet, which may contain a subset $C$ of constants, to which no rules will apply,
- $R$ is a set of rules,
- $\omega$ (the start sequence) is an element of $\Sigma^+$.

Each rule in $R$ is of the form: $\alpha A \beta \rightarrow \gamma$, where:

- $A \in \Sigma$. $A$ is the symbol that is to be rewritten by the rule.
- $\alpha, \beta \in \Sigma^*$. $\alpha$ and $\beta$ describe context that must be present in order for the rule to fire. If they are equal to $\varepsilon$, no context is checked.
- $\gamma \in \Sigma^*$. $\gamma$ is the string that will replace $A$ when the rule fires.
Rules Fire in Parallel

Using a standard grammar:

Given: \[ A \ S \ a \ B \ B \ a \]

\[ \downarrow \]

\[ B \ S \ a \ B \ B \ a \]

\[ \downarrow \]

\[ B \ S \ a \ C a \ B \ a \]

etc.

Using an L-system:

Given: \[ A \ S \ a \ B \ B \ a \]

\[ \downarrow \ \downarrow \ \downarrow \ \downarrow \ \downarrow \]

\[ B \ F \ a \ H a \ H a \ a \]
A Lindenmayer System Interpreter

$L$-system-interpret($G$: L-system) =
1. Set working-string to $\omega$.
2. Do forever:
   2.1 Output working-string.
   2.2 new-working-string = $\varepsilon$.
   2.3 For each character $c$ in working-string do:
      If possible, choose a rule $r$ whose left-hand side matches $c$ and where $c$’s neighbors (in working-string) satisfy any context constraints included in $r$.
      If a rule $r$ was found, concatenate its right-hand side to the right end of new-working-string.
      If none was found, concatenate $c$ to the right end of new-working-string.
   2.4 working-string = new-working-string.
An Example

Let $G$ be the L-system defined as follows:

$$\Sigma = \{I, M\}.$$
$$\omega = I.$$
$$R = \{I \rightarrow M,\quad M \rightarrow M\ I\}.$$ 

The sequence of strings generated by $G$ begins:

0.  $I$
1.  $M$
2.  $M\ I$
3.  $M\ I\ M$
4.  $M\ I\ M\ M\ I$
5.  $M\ I\ M\ M\ I\ M\ I\ M$
6.  $M\ I\ M\ M\ I\ M\ I\ M\ M\ I\ M\ M\ I$
Fibonacci’s Rabbits

Assume:
- It takes one time step for each rabbit to reach maturity and mate.
- The gestation period of rabbits is one time step.
- We begin with one pair of (immature) rabbits.

The Fibonacci sequence:

\[
\begin{align*}
Fibonacci_0 &= 1, \\
Fibonacci_1 &= 1, \\
\text{For } n > 1, 
Fibonacci_n &= Fibonacci_{n-1} + Fibonacci_{n-2}.
\end{align*}
\]

The L-system model:

- Each \( \mathcal{I} \) corresponds to one immature pair of rabbits.
- Each \( \mathcal{M} \) corresponds to one mature pair.
- Each string is the concatenation of its immediately preceding string (the survivors) with the string that preceded it two steps back (the breeders).
Fibonacci’s Rabbits

0. I
1. M
2. MI
3. MIM
4. MIMMI
5. MIMMIMIM
6. MIMMIMIMIMIMIMIM
Sierpinski Triangle

Let $G$ be the L-system defined as follows:

$$\Sigma = \{A, B, +, -\}.$$  
$$\omega = A.$$  
$$R = \{ A \rightarrow B - A - B, \quad B \rightarrow A + B + A \}.$$  

Notice that $+$ and $-$ are constants. The sequence of strings generated by $G$ begins:

1. $A$
2. $B - A - B$
3. $A + B + B - A - B - A + B + A$
Interpreting Strings as Drawing Programs

- Choose a line length $k$.
- Attach meanings to the symbols in $\Sigma$ as follows:
  - $A$ and $B$ mean move forward, drawing a line of length $k$.
  - $+$ means turn to the left $60^\circ$.
  - $-$ means turn to the right $60^\circ$.

Strings 3, 4, 8, and 10 then correspond to turtle programs that can draw the following sequence of figures (scaling $k$ appropriately):
Sierpinski Triangles
Modelling Plant Growth

Let $G$ be the L-system defined as follows:

$$\Sigma = \{F, +, –, [, ]\}.$$
$$\omega = F.$$
$$R = \{F \rightarrow F [ – F ] F [ + F ] [ F ]\}.$$

The sequence of strings generated by $G$ begins:

1. $F$
2. $F [ – F ] F [ + F ] [ F ]$
   $F [ – F ] F [ + F ] [ F ] [ + F [ – F ] F$
   $[ + F ] [ F ] ] [ F [ – F ] F [ + F ] [ F ] ]$
Interpreting Strings as Drawing Programs

We can interpret these strings as turtle programs by choosing a line length $k$ and then attaching meanings to the symbols in $\Sigma$ as follows:

- F means move forward, drawing a line of length $k$.
- + means turn to the left $36^\circ$.
- – means turn to the right $36^\circ$.
- [ means push the current pen position and direction onto the stack.
- ] means pop the top pen position/direction off the stack, lift up the pen, move it to the position that is now on the top of the stack, put it back down, and set its direction to the one on the top of the stack.

   $F [- F] F [+ F] [F] [+ F [- F] F$
   $[+ F] [F] [F [- F] F [+ F] [F] ]$
Trees
Trees
Real Plants

http://algorithmicbotany.org/papers/#abop
Sierpinski Triangles, Again

Let $G$ be the L-system defined as follows:

$\Sigma = \{\blacksquare, \square\}$.

$\omega = \blacksquare$.

$R = \{ (\varepsilon \mid \square) \blacksquare (\varepsilon) \rightarrow \blacksquare \blacksquare,$

$(\varepsilon \mid \square) \blacksquare (\blacksquare \mid \square) \rightarrow \blacksquare,$

$(\blacksquare) \blacksquare (\varepsilon) \rightarrow \square \blacksquare,$

$(\blacksquare) \blacksquare (\blacksquare \mid \square) \rightarrow \square,$

$(\varepsilon \mid \square) \square (\varepsilon) \rightarrow \square \blacksquare,$

$(\varepsilon \mid \square) \square (\blacksquare \mid \square) \rightarrow \square,$

$(\blacksquare) \square (\varepsilon) \rightarrow \blacksquare \blacksquare,$

$(\blacksquare) \square (\blacksquare \mid \square) \rightarrow \blacksquare \}.$
Equivalence of Context-Sensitive L-Systems and Turing Machines

*Theorem:* Context-sensitive L-systems and Turing machines can simulate each other.

*Proof:* The computation of any L-system can be simulated by a Turing machine that implements the algorithm *L-system-interpret*. So it remains to show the other direction, by construction.

We show that deterministic TM \( M \), on input \( w \), halts in some halting state \( q \) and with tape contents \( v \) iff L-system \( L \) converges to the static string \( qv \).

Define \( L \) as follows:

- Let \( \Sigma_L \) be \( \Sigma_M \), augmented by adding the symbols \( y, n, \) and \( h \), plus one distinct symbol for each nonhalting state of \( M \). Assume that \( 0 \) encodes \( M \)'s start state.
Defining $L$, Continued

Let $\omega$ encode $M'$'s initial configuration as:

\[ \square \omega \square \square \omega \square . \]

Let the rules of $L$ encode $M'$'s transitions. The action of a Turing machine is very local. Consider:

\[ ga4bcde. \]

The read/write head can move one square and the character under it can change. To decide how to rewrite some character in the working string, it is sufficient to look at one character to its left and two to its right. If there is no state symbol in that area, the symbol gets rewritten as itself (no rule needed). For all other combinations, add to $R$ rules that cause the system to behave as $M$ behaves. Finally, add rules so that, if $h$, $y$, or $n$ is generated, it will be pushed all the way to the left, leaving the rest of the string unchanged. Add no other rules to $R$.

$L$ will converge to $qv$ iff $M$ halts, in state $q$, with $v$ on its tape.