Grammars

An unrestricted grammar is formally described by a four-tuple $G = (V, \Sigma, R, S)$ of a finite set of variables $V$, a finite set of alphabet symbols $\Sigma$, a finite set of grammar rules $R$, and a start symbol $S \in V$. All are similar to context-free grammars except that the rules are allowed to have more than one symbol on the left-hand side as long as it has at least one variable. Thus, we require

$$R \subseteq (V \cup \Sigma)^*V(V \cup \Sigma)^* \times (V \cup \Sigma)^*.$$ 

Example A grammar $G$ generating $L = \{ww : w \in \{a, b\}^*\}$ where $V = \{S, A, B, [, ]\}$, $\Sigma = \{a, b\}$, and $R$ is the following set of rules:

$$
\begin{align*}
S & \rightarrow A \\
A & \rightarrow aAa \mid bAb \\
[a] & \rightarrow [A] \\
[b] & \rightarrow [B] \\
Aa & \rightarrow aA \\
Ab & \rightarrow bA \\
Ba & \rightarrow aB \\
Bb & \rightarrow bB \\
A] & \rightarrow ]a \\
B] & \rightarrow ]b \\
[ & \rightarrow \epsilon
\end{align*}
$$

The idea of the grammar is to generate $w[w^R]$ and then reverse $w^R$. Reversing $w^R$ is achieved by allowing the left-marker $[\ ]$ to create variable copies of the terminal symbols and allow them to migrate from left to right. Once these variables arrived and encounter the right-marker $]$, they turn themselves back to terminals. The order in which these symbols migrate will cause the string $w^R$ to be reversed in the end. At the end, when the two left and right markers meet, they disappear together and leave the string $ww$ as the result.
Machines

A Turing machine is formally defined as a four-tuple $M = (Q, \Sigma, \delta, s)$ consisting of a finite set of states $Q$, a finite alphabet $\Sigma$, a transition function $\delta : Q \times \Sigma \rightarrow (Q \cup \{h\}) \times \Sigma \times \{L, R\}$, and a start state $s \in Q$. We assume the halt state $h$ is not part of $Q$ but the blank symbol $\Box$ is an element of $\Sigma$.

We describe further aspects of the model:

- The transition $\delta(q, a) = (p, b, \tau)$ encodes the condition that: when $M$ is in state $q$ while reading $a \in \Sigma$ on the input tape, it will switch to state $p$, writing $b$ in place of $a$, and moving the tape reader in the direction specified by $\tau \in \{L, R\}$.
- $M$ has access to a one-way infinite tape (which we may think as indexed by the natural numbers). Initially, the tape contains a finite number of non-blank symbols. Warning: If the tape reader is moved left off the square indexed 0, then $M$ hangs indefinitely and may require rebooting.
- A configuration of $M$ is given by a four-tuple $(q, \alpha, a, \beta)$ where $q$ denotes the current state and the contents of the tape is $\alpha a \beta$, where $\alpha \in \Sigma^*$ is the tape contents to the left of the tape reader, $a \in \Sigma$ is the symbol currently being read by the tape reader, and $\beta \in \Sigma^*$ is the tape contents to the right of the tape reader (the last symbol of $\beta$ is the rightmost non-blank symbol on the tape). Sometimes, the more cryptic notation $(q, \alpha a \beta)$ is used in place of $(q, \alpha, a, \beta)$. A configuration whose state is $h$ is called a halted configuration.

As with the simpler models, we define the yield-in-one-step relation $\vdash_M$ for a Turing machine $M$. We say

\[ (q_1, \alpha_1 a_1 \omega_1) \vdash_M (q_2, \alpha_2 a_2 \omega_2) \]

iff either

- $\delta(q_1, a_1) = (q_2, b, L)$, $\alpha_1 = \alpha_2 a_2$, and $\omega_2 = b \omega_1$; or
- $\delta(q_1, a_1) = (q_2, b, R)$, $\alpha_2 = \alpha_1 b$, and $\omega_1 = a_2 \omega_2$.

Now, we can talk about the transitive closure of $\vdash_M^*$. We say $M$ halts on $\alpha$ if

\[ (s, \Box \alpha \Box) \vdash_M^* (h, \beta \alpha \gamma) \]

for some $\beta, \gamma \in \Sigma^*$ and $a \in \Sigma$. 2
• A Turing machine $M = (Q, \Sigma, \delta, s)$ computes a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$, where $\Sigma_1, \Sigma_2 \subseteq \Sigma$ are alphabets (not containing blanks), if

$$(s, \square \alpha \square) \vdash^*_M (h, \square \beta \square),$$

where $\beta = f(\alpha)$. In such case, we say $f$ is a **Turing-computable** function.

• We may also use Turing machines as language recognizers (much like DFAs and PDAs) as follows. We augment the set of states $Q$ with two additional (halting) states called $y$ and $n$ (distinct from the halt state $h$).

A Turing machine $M$ **accepts** a string $\alpha \in \Sigma^*$ if on input $\alpha$, $M$ enters the state $y$ (and halts). Moreover, we say $M$ **accepts** a language $L$ if $M$ accepts $\alpha$ for all $\alpha \in L$. Note, we do not require $M$ to halt on inputs that are not in $L$. The language $L(M)$ accepted by a Turing machine $M$ is then

$$L(M) = \{ \alpha \in \Sigma_1^* : M \text{ accepts } \alpha \}.$$

Thus, a language $L$ is **Turing-acceptable** if there is a Turing machine that accepts it. A Turing-acceptable language is also called a semi-decidable or recursively enumerable language.

A Turing machine $M$ **decides** a language $L$ if for all strings $\alpha \in \Sigma^*$:

$$M(\alpha) = \begin{cases} 
\text{enters } y & \text{if } \alpha \in L \\
\text{enters } n & \text{if } \alpha \not\in L
\end{cases}$$

Note, for deciding, we require that $M$ halts and enters one of the two state $y$ or $n$ on all possible inputs (unlike the one-sided requirement for accepting).

In this case, $L$ is called a **Turing-decidable** language. A Turing-decidable (or simply decidable) language is also called a recursive language.

**Examples**

1. A Turing machine to **compute** $f(0^m) = 0^{m+1}$, for $m \geq 0$.

![Figure 1: A Turing machine for incrementing a unary number.](image)

2. A Turing machine to **decide** if its input (binary number) is even or not.
Variants

We consider different variants of the basic Turing machine model.

1. A Turing machine with **two-way infinite tape** (which we may think as indexed by the integers).

2. A **k-tape** Turing machine (where each tape is one-way infinite), for \( k \geq 2 \).

3. A Turing machine with **k tape-readers**, for \( k \geq 2 \).

4. A Turing machine with **k-dimensional** tape, for \( k \geq 2 \).

5. A **non-deterministic** Turing machine, where \( \delta \) is a transition relation.

These different models can be shown to be equivalent to the basic model.

Equivalence

**Theorem 1.** If \( L \) is a language generated by a grammar \( G \), there is a Turing machine \( M \) that accepts \( L \).

**Theorem 2.** If \( L \) is a Turing-acceptable language, there is a grammar \( G \) that generates \( L \).
Closure Properties

1. If $L$ is recursive, then so is $L^c$.
2. If $L_1$ and $L_2$ are recursive, then so is $L_1 \cup L_2$.
3. If $L_1$ and $L_2$ are recursively enumerable, then so is $L_1 \cup L_2$.
4. If $L$ and $L^c$ are both recursively enumerable, then $L$ is recursive.

Undecidability

By a counting argument (or a disguised pigeonhole principle), we may establish the existence of an undecidable language. This is because the number of languages over a fixed alphabet $\Sigma$ is not countable, but there are only countably many Turing machines (since it has a finite description). But, there are more natural examples of undecidable languages.

We fix an encoding of Turing machines and use $\langle M \rangle$ to denote the encoding of a Turing machine $M$.

**Theorem 3.** The language $\text{HALT} = \{\langle M \rangle, \alpha \mid M \text{ halts on } \alpha\}$ is undecidable.

**Proof.** Assume $\text{HALT}$ is decidable by a Turing machine $U$. We define another Turing machine called $D$ where

$$D(\langle M \rangle) = \begin{cases} \text{halt} & \text{if } U(\langle M \rangle, \langle M \rangle) = \bot \\ \text{run forever} & \text{if } U(\langle M \rangle, \langle M \rangle) = \top \end{cases}$$

We arrive at a contradiction by asking if $D(\langle D \rangle)$ halts or not. □

Next, we show a reduction from $\text{HALT}$ to $\text{EPSILON}$, or in notation, $\text{HALT} \preceq \text{EPSILON}$. This means that if there is a decider for $\text{EmptyString}$, then there is a decider for $\text{HALT}$. Since we know that there is no decider for $\text{HALT}$, the decider for $\text{EPSILON}$ does not exist.

**Theorem 4.** The language $\text{EPSILON} = \{\langle M \rangle \mid M \text{ halts on } \epsilon\}$ is undecidable.

**Proof.** Assume that $\text{EPSILON}$ is decidable by a Turing machine $E$. Consider the following Turing machine $U$:

$$U(\langle M \rangle, \alpha):$$

define a Turing machine $D$ where:
\[
D(x): \\
\text{if } x = \epsilon \text{ then run } M(\alpha) \text{ else run forever}
\]

run \(E(\langle D \rangle)\)

Note \(D\) halts on \(\epsilon\) iff \(M\) halts on \(\alpha\). So, \(U\) decides \text{HALT}, a contradiction. \(\square\)

Let \(\Psi\) be a \textit{property} of recursively enumerable languages. Here, we simply mean that \(\Psi\) a collection recursively enumerable languages. The property \(\Psi\) is called non-trivial if \(\Psi\) is neither empty nor contain all recursively enumerable languages. Let \(L_\Psi = \{\langle M \rangle : L(M) \in \Psi\}\) be the set of recursively enumerable languages with property \(\Psi\).

\textbf{Theorem 5.} (\textit{Rice’s Theorem}) \textit{For any non-trivial property} \(\Psi\), \(L_\Psi\) is undecidable.

\textit{Proof.} Assume \(\emptyset \notin \Psi\) (otherwise we may use \(\Psi^c\) instead). Since \(\Psi\) is non-trivial, there is a language \(\hat{L} \in \Psi\). Suppose \(\hat{M}\) is a Turing machine that accepts \(\hat{L}\).

For the purpose of contradiction, we suppose that \(L_\Psi\) is decidable and let \(M_\Psi\) be a Turing machine that decides \(L_\Psi\). Consider the following Turing machine \(U\):

\[
U(\langle M \rangle, \alpha): \\
\text{create a Turing machine } N: \\
N(x) = \{\text{if } M(\alpha) \text{ accepts then run } \hat{M}(x)\} \\
\text{run } M_\Psi(\langle N \rangle)
\]

Note \(L(N) \in \Psi\) iff \(M\) accepts \(\alpha\). This shows \text{HALT} is decidable by \(U\). \(\square\)