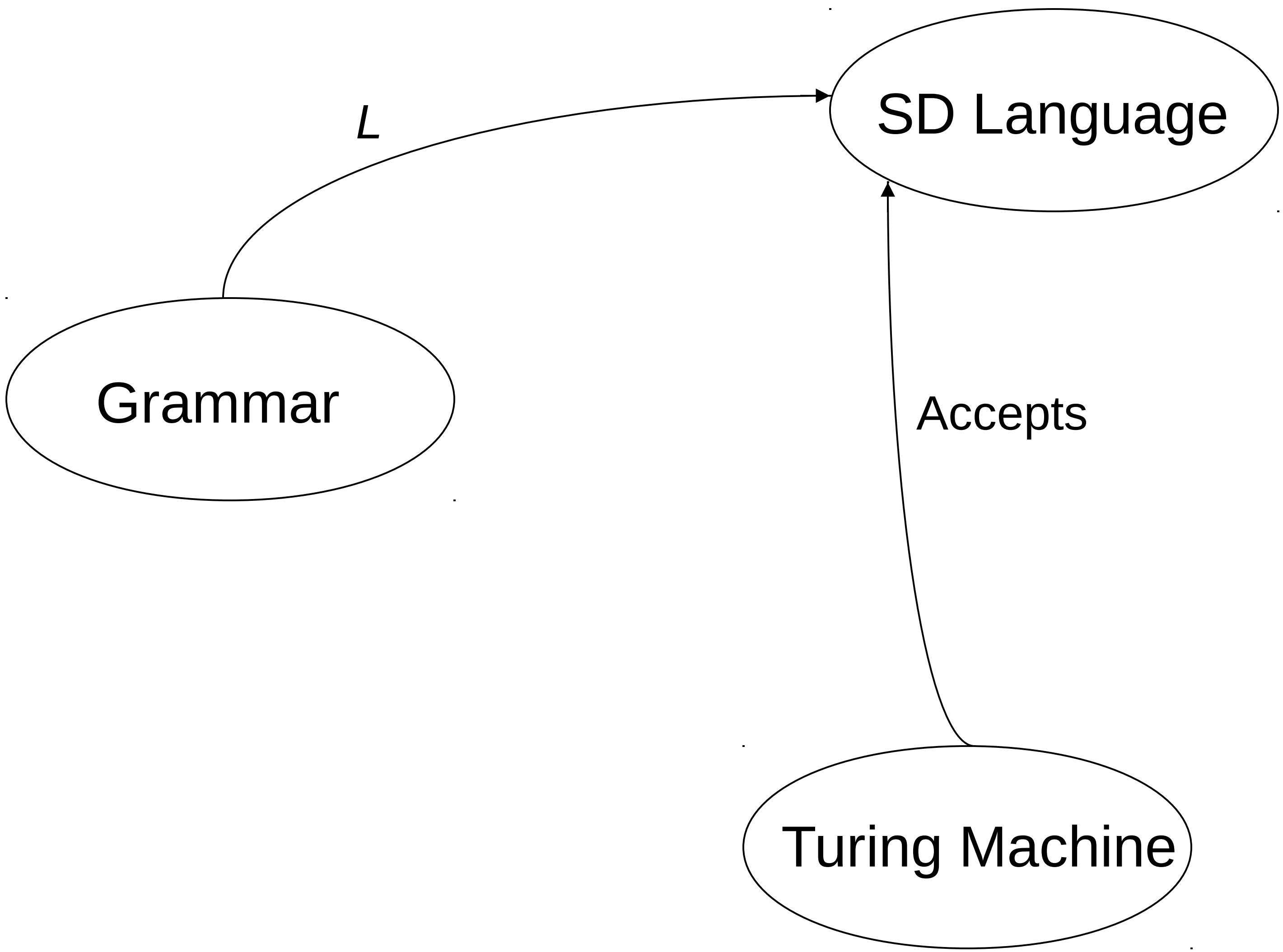
Unrestricted Grammars

Chapter 23

Grammars, SD Languages, and Turing Machines



Unrestricted Grammars

An ***unrestricted grammar*** G is a quadruple (V, Σ, R, S), where:

- V is an alphabet,
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V^+ \times V^*)$,
- S (the start symbol) is an element of $V - \Sigma$.

The language generated by G is:

$$\{w \in \Sigma^* : S \Rightarrow_G^* w\}.$$

Unrestricted Grammars

Example: $A^nB^nC^n = \{a^n b^n c^n, n \geq 0\}$.

$$S \rightarrow aBSc$$
$$S \rightarrow \varepsilon$$
$$Ba \rightarrow aB$$
$$Bc \rightarrow bc$$
$$Bb \rightarrow bb$$

Proof:

- Only strings in $A^nB^nC^n$:

- All strings in $A^nB^nC^n$:

Another Example

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

$$
\begin{aligned}
&S \to ABCS \\
&S \to \varepsilon \\
&AB \to BA \\
&BA \to AB \\
&BC \to CB \\
&CB \to BC \\
&AC \to CA \\
&CA \to AC \\
&A \to \mathrm{a} \\
&B \to \mathrm{b} \\
&C \to \mathrm{c}
\end{aligned}
$$

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

Idea:

1. Generate a string in ww^R, plus delimiters

aaabb*C*bbaaa#

2. Reverse the second half.

WW = {*ww* : *w* ∈ {a, b}*}

$S \rightarrow T\#$	/* Generate the wall exactly once.
$T \rightarrow \mathrm{a}T\mathrm{a}$	/* Generate wCw^R.
$T \rightarrow \mathrm{b}T\mathrm{b}$	″
$T \rightarrow C$	″
$C \rightarrow CP$	/* Generate a pusher P
$P\mathrm{aa} \rightarrow \mathrm{a}P\mathrm{a}$	/* Push one character to the right to get ready to jump.
$P\mathrm{ab} \rightarrow \mathrm{b}P\mathrm{a}$	
$P\mathrm{ba} \rightarrow \mathrm{a}P\mathrm{b}$	″
$P\mathrm{bb} \rightarrow \mathrm{b}P\mathrm{b}$	″
$P\mathrm{a}\# \rightarrow \#\mathrm{a}$	/* Hop a character over the wall.
$P\mathrm{b}\# \rightarrow \#\mathrm{b}$	″
$C\# \rightarrow \varepsilon$	

Equivalence of Unrestricted Grammars and Turing Machines

Theorem: A language is generated by an unrestricted grammar if and only if it is in SD.

Proof:

Only if (grammar → TM): by construction of an NDTM.

If (TM → grammar): by construction of a grammar that mimics the behavior of a semideciding TM.

Grammar → Turing Machine

Given *G*, produce a Turing machine *M* that semidecides *L*(*G*).

M will be nondeterministic and will use two tapes:

	❑	a	b	a	b	❑	❑	
❑	1	0	0	0	0	0	0	❑
	a	*S*	*T*	a	a	b	❑	
	1	0	0	0	0	0	0	

For each nondeterministic "incarnation":

- Tape 1 holds the input.
- Tape 2 holds the current state of a proposed derivation.

At each step, *M* nondeterministically chooses a rule to try to apply and a position on tape 2 to start looking for the left-hand side of the rule. Or it chooses to check whether tape 2 equals tape 1. If any such machine succeeds, we accept. Otherwise, we keep looking.

Turing Machine → Grammar

Build G to simulate the forward operation of a TM M:

The first (**generate**) part of G:
Create all strings over Σ* of the form:

w = # ❑ ❑ q000 a_1 a_1 a_2 a_2 a_3 a_3 ❑ ❑ #

(Blue copy to store for later, red copy to simulate M on.)

The second (**test**) part of G simulates the execution of M on a particular string w (the red copy). An example of a partially derived string:

❑ ❑ a 1 b 2 c c b 4 q001 a 3

Examples of rules:

q100 b b → b 2 q101

a a q011 b 4 → q011 a a b 4

The Last Step

The third (**cleanup**) part of G erases the working part if M ever reaches any of its accepting states, all of which will be encoded as A. The saved input string is then generated by the grammar (because it was accepted by M).

Rules:

$\forall z$ z A $\rightarrow$ A z /* Sweep A to the left.

$\forall x, y$ #A x y $\rightarrow$ x #A /* Erase working part

#A# $\rightarrow$ ε

Decision Problems for Unrestricted Grammars

- Given a grammar G and a string w, is $w \in L(G)$?
- Given a grammar G, is $\varepsilon \in L(G)$?
- Given two grammars G_1 and G_2, is $L(G_1) = L(G_2)$?
- Given a grammar G, is $L(G) = \varnothing$?

Or, as languages:

- $L_a = \{<G, w> : w \in L(G)\}$.
- $L_\varepsilon = \{<G> : \varepsilon \in L(G)\}$.
- $L_= = \{<G_1, G_2> : L(G_1) = L(G_2)\}$.
- $L_\varnothing = \{<G> : L(G) = \varnothing\}$.

None of these questions is decidable.

$L_a = \{<G, w> : w \in L(G)\}$ is not in D.

Proof: Let R be a mapping reduction from:

A = $\{<M, w>$: Turing machine M accepts $w\}$ to L_a:

$R(<M, w>)$ =

1. From M, construct the description $<G\#>$ of a grammar $G\#$ such that $L(G\#) = L(M)$.
2. Return $<G\#, w>$.

If *Oracle* decides L_a, then C = *Oracle*($R(<M, w>)$) decides A. We have already defined an algorithm that implements R. C is correct:

- If $<M, w> \in$ A : $M(w)$ halts and accepts. $w \in L(M)$. So $w \in L(G\#)$. *Oracle*($<G\#, w>$) accepts.
- If $<M, w> \notin$ A : $M(w)$ does not accept. $w \notin L(M)$. So $w \notin L(G\#)$. *Oracle*($<G\#, w>$) rejects.

But no machine to decide A can exist, so neither does *Oracle*.