Context-Free Languages

CSCI 2210

2023-10-16 and 2023-10-18

Non-Regular Languages

We used the *pumping lemma* to show that some string languages are not regular.

Among these are

- $0^n\,1^n$, the language of a run of 0s followed by an equal-length run of 1s,
- $w\,w^{\mathrm{R}}$, the language of even-length palindromes (strings followed by their own reversal).

Intuitively, this is because we need an unbounded amount of *memory* to keep track of some feature of interest.

But the amount of information that an NFA can store is bounded by the number of its states.

Adding Memory to an NFA

We can remedy this by giving an NFA an external memory.

There are several possible *data structures* we could use for such a memory:

dictionary: providing *random access*

linked list: providing *sequential access*

tree: providing *hierarchical access*

etc.

Stack Memory

Here, we consider a very simple form of memory in the form of a stack.

A **stack** is either **empty** or else consists of a **head** element and a **tail** stack.

Given any element x and stack xs we can **push** x onto xs forming a stack with head x and tail xs.

If xs is a nonempty stack then we can **pop** it to obtain its head element and tail stack.

For any element and stack we have that:

$$\mathsf{pop}(\mathsf{push}(x\ ,\ xs)) = (x\ ,\ xs)$$

i.e.

$$\mathsf{push} \cdot \mathsf{pop} = \mathrm{id}$$

Pushdown Automata

A (nondeterministic ε-) **pushdown automaton** ("PDA") M has the following components:

input alphabet: a nonempty finite set of symbols Σ,

state set: a nonempty finite set of states Q,

start state: a chosen state $q_0 \in \mathrm{Q}$,

accept state set: a chosen subset of states $\mathrm{F} \subseteq \mathrm{Q}$,

stack alphabet: a finite set of symbols Γ,

state transition relation: a relation $\Delta : \mathrm{Q} \times \Sigma_\varepsilon \times \Gamma_\varepsilon \nrightarrow \mathrm{Q} \times \Gamma_\varepsilon$.

PDA State Transition Graph

We can represent a PDA M using a state transition graph.

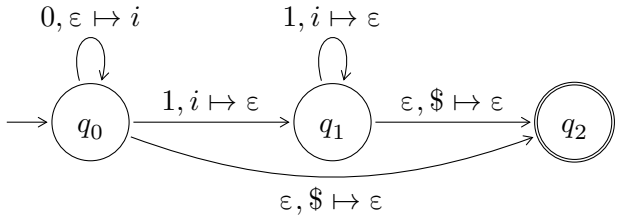
The vertex set is Q.

For vertices $q_a, q_b \in \mathrm{Q}$, optional input symbol $s \in \Sigma_\varepsilon$, and optional stack symbols $x, y \in \Gamma_\varepsilon$, there is an edge $(s, x \mapsto y) : q_a \to q_b$ just in case $\Delta\left((q_a\ ,\ s\ ,\ x) \dashrightarrow (q_b\ ,\ y)\right)$.

- The notation "$x \mapsto y$" means that we pop x and push y on the stack.
- If x resp. y is ε then we don't pop resp. push anything.
- We use "$" as shorthand for empty stack, so "$\$ \mapsto y$" means "push y to the empty stack".

Example: PDA State Transition Graph Diagram

Let $\mathrm{M}_{0^n1^n}$ be the PDA specified by:



This means:

- $\Sigma = \{0, 1\}$,
- $\mathrm{Q} = \{q_0, q_1, q_2\}$,
- $q_0 = q_0$,
- $\mathrm{F} = \{q_2\}$,
- $\Gamma = \{i\}$,

Δ	0			1			ε		
	i	\$	ε	i	\$	ε	i	\$	ε
q_0			$\{(q_0\,, i)\}$	$\{(q_1\,, \varepsilon)\}$				$\{(q_2\,, \varepsilon)\}$	
q_1				$\{(q_1\,, \varepsilon)\}$				$\{(q_2\,, \varepsilon)\}$	
q_2									

Language of a PDA: Operational Semantics

Each PDA M over input alphabet Σ determines a **language** $\mathrm{L}(\mathrm{M})$ according to the following **operational semantics**.

Given an input string $w \in \Sigma^*$:

- M begins in the start state q_0 with an empty stack \$,
- M inspects the symbols of w left-to-right one at a time,
- If M is in state q_a inspecting input symbol s then M *may* transition to state q_b if $\Delta\left((q_a\ ,\ s\ ,\ x)\right.$ -↠ $\left.(q_b\ ,\ y)\right)$ by popping x and pushing y on the stack,
- If x rsp. y is ε then M may transition without popping rsp. pushing the stack,
- If $\Delta\left((q_a\ ,\ \varepsilon\ ,\ x)\right.$ -↠ $\left.(q_b\ ,\ y)\right)$ then M may transition without consuming input,
- Upon exhausting w, there is a *set of states* Q_f that M may be in,
- We say $w \in \mathrm{L}(\mathrm{M})$ just in case $\mathrm{Q}_f \cap \mathrm{F} \neq \emptyset$.

Language of a PDA: State Transition Graph Interpretation

For a PDA M, each word $w \in \Sigma^*$ determines a *set of paths* in the state transition graph all with source q_0.

The state transition *relation* Δ determines where these paths go at each step.

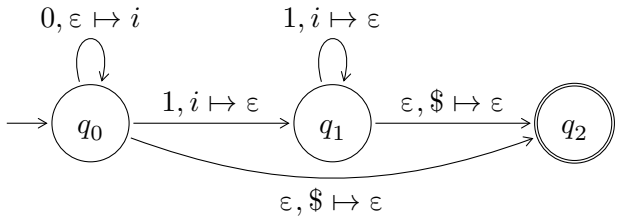
We can extend a path $p : q_0 \to q_a$ by an edge $(s\ ,\ x \mapsto y) : q_a \to q_b$ if the next symbol in the word is s, the top of the stack is x, and $\Delta\ ((q_a\ ,\ s\ ,\ x) \dashrightarrow (q_b\ ,\ y))$, or if s resp. x is ε without consulting the word resp. stack.

We take the *union of all paths* obtained by extending each path p by each such edge.

A path $q_0 \to q_f$ constructed in this way is called a **run** of M.

PDA M accepts word w just in case there is an accepting run of M labeled by w.

Example: Language of $\mathrm{M}_{0^n1^n}$



decides the language of words with a run of 0s followed by an equal-length run of 1s.

for example:

- word $0\,1$ is accepted by run $[(q_0), 0, (q_0), 1, (q_1), \varepsilon, (q_2)]$
- word $0\,0\,1\,1$ is accepted by run $[(q_0), 0, (q_0), 0, (q_0), 1, (q_1), 1, (q_1), \varepsilon, (q_2)]$

We have left the effect on the stack implicit, but it must be verified.

Context-Free Languages

We call a string language **context-free** if there is a PDA that decides it.

Theorem

Every *regular* string language is *context-free*.

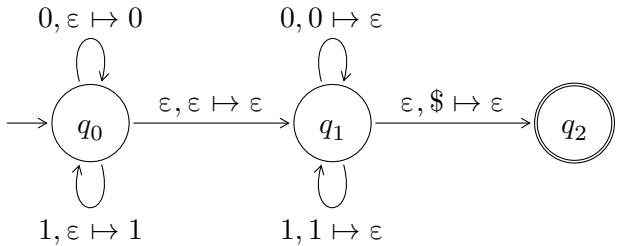
Proof.

An ε-NFA is just a PDA without a stack.

So a regular language is decided by a PDA with stack alphabet $\Gamma := \emptyset$. □

Activity: Language of a PDA

Give a description of the language decided by the following PDA:



Machine Models and Language-Based Models

Recall that we have two equivalent ways to describe the *regular* string languages:

finite automata: a family of machine models which *decides* them,

regular expressions: a language-based model which *describes* them.

We saw last time that PDAs constitute a machine model for deciding *context-free* string languages.

In fact, there is a language-based model for describing them as well.

Context-Free Grammars

A **context-free grammar** (“CFG”) consists of:

terminals: a finite set of symbols Σ

nonterminals: a disjoint nonempty finite set of symbols V

start symbol: a symbol $\mathrm{S} \in \mathrm{V}$

production rules: a relation $\mathrm{R} : \mathrm{V} \nrightarrow (\mathrm{V} \cup \Sigma)^*$

The nonterminals are also called “variables”.

The relation R can be expressed as a list of rules, called “production clauses”.

By convention the source of the first clause is the start symbol S.

Example: Context-Free Grammar $\mathrm{G}_{0^n1^n}$

Consider the context-free grammar given by:

terminals: $\Sigma := \{0, 1\}$

nonterminals: $\mathrm{V} := \{\mathrm{A}\}$

start symbol: A

production rules: $\mathrm{R} :=$

r_1: $\mathrm{A} \dashrightarrow 0\,\mathrm{A}\,1$

r_2: $\mathrm{A} \dashrightarrow \varepsilon$

Parse Tree of a Context-Free Grammar

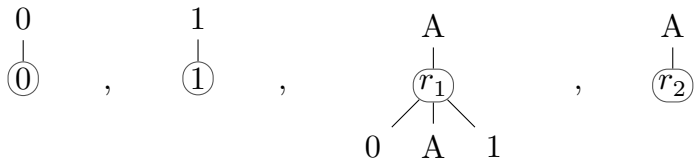
A context-free grammar G determines a set of **parse trees** as follows:

- Each terminal and nonterminal symbol determines a label for edges.
- Each *terminal symbol* $t \in \Sigma$ also determines a **leaf node** whose incoming edge is labeled by t.
- Each *production clause* $r : \mathrm{A} \dashrightarrow w$ determines an **internal node** whose incoming edge is labeled by A and whose outgoing edges are labeled by w.

Any tree with root labeled by the start symbol built in this way is a parse tree for G.

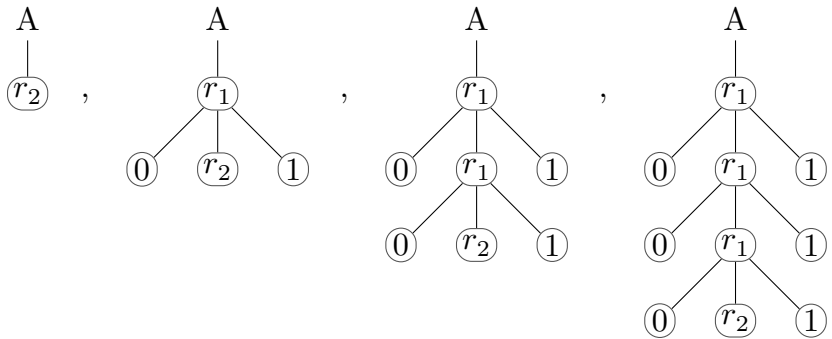
Example: Parse Trees for $\mathrm{G}_{0^n1^n}$

The nodes for parse trees of the context free grammar $\mathrm{G}_{0^n1^n}$ are:



Example: Parse Trees for $\mathrm{G}_{0^n1^n}$ ctd.

From these we can build parse trees such as:

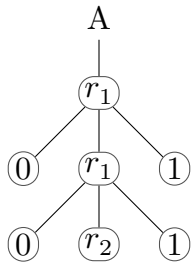


We may omit writing either the edge labels or the clause labels,
as each can be inferred from the other.

Tree Fringe

The **fringe** of a tree is the finite sequence of its *leaf nodes*, in the same order they appear in the tree.

For example, the fringe of



is $[0, 0, 1, 1]$.

Language of a Context-Free Grammar

The fringe of a parse tree for a context-free grammar G is a word over the alphabet of its terminal symbols.

The set of all such words is the (string) **language** of this context-free grammar, $\mathrm{L}(\mathrm{G})$.

For example, $0\,0\,1\,1 \in \mathrm{L}(\mathrm{G}_{0^n1^n})$.

Indeed, $\mathrm{L}(\mathrm{G}_{0^n1^n})$ is the string language of words consisting of a run of 0s followed by an equal-length run of 1s.

We used the pumping lemma to show that this language is not regular,
and we built a PDA that decides it.

Activity: CFG for Palindromes

Devise a context-free grammar $\mathrm{G}_{ww^{\mathrm{R}}}$ so that $\mathrm{L}(\mathrm{G}_{ww^{\mathrm{R}}})$ is $\mathrm{L}_{ww^{\mathrm{R}}}$,
the string language of even-length palindromes over the alphabet $\{0, 1\}$.

Then test your grammar by making parse trees for:

- ε
- $0\,1\,1\,0$
- $1\,1\,0\,0\,1\,1$

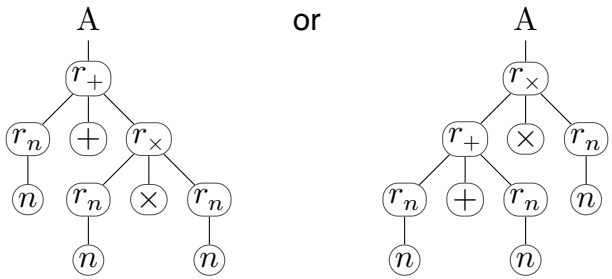
Ambiguity

Some CFGs allow us to construct more than one parse tree for a given string.

Consider the CFG implied by:

$$r_+ : \mathrm{A} \dashrightarrow \mathrm{A} + \mathrm{A} \quad , \quad r_\times : \mathrm{A} \dashrightarrow \mathrm{A} \times \mathrm{A} \quad , \quad r_n : \mathrm{A} \dashrightarrow n$$

The string $n + n \times n$ could be the fringe of either of the following parse trees:



Ambiguous Grammars

CFGs where a string can have multiple possible parse trees are **ambiguous**.

Often it is possible to convert an ambiguous CFG into an unambiguous one that specifies the same language.

However, there are some string languages that have only ambiguous CFGs.

Some Intuition for CFGs

Each nonterminal symbol of a CFG determines a tree-structured language.

The way these nonterminals appear in the production rules for each other determines the mutually inductive structure of these languages.

CFGs were originally proposed as a way to model the recursive structure of natural languages like English.

They have since become invaluable in parsing for programming languages.

A Natural Language CFG

SENTENCE	→	NOUN-PHRASE VERB-PHRASE
NOUN-PHRASE	→	CMPLX-NOUN | CMPLX-NOUN PREP-PHRASE
VERB-PHRASE	→	CMPLX-VERB | CMPLX-VERB PREP-PHRASE
PREP-PHRASE	→	PREP CMPLX-NOUN
CMPLX-NOUN	→	ARTICLE NOUN
CMPLX-VERB	→	VERB | VERB NOUN-PHRASE
ARTICLE	→	a | the
NOUN	→	boy | girl | flower
VERB	→	sees | likes
PREP	→	with

Equivalence of PDAs and CFGs

Theorem

A string language is decided by a PDA just in case it is described by a CFG.