Griffith University

3515ICT Theory of Computation

Context-Free Languages

(Based loosely on slides by Harald Søndergaard of The University of Melbourne)

Context-Free Grammars

. . . were invented in the fifties, when Chomsky proposed different formalisms for describing natural language syntax.

They were popularised by Naur with the Algol 60 report, and programming language grammars are sometimes presented in this Backus-Naur Form (BNF).

Standard tools for parsing owe much to this notation, which has helped make parsing a routine task.

Context-free grammars are extensively used to specify the syntax of programming languages, and now the structure of documents (XML's document-type definitions).

Context-Free Grammars (cont.)

We can specify the syntax (or form) of regular expressions with the following grammar:

$$\begin{aligned} R &\rightarrow 0 \\ R &\rightarrow 1 \\ R &\rightarrow \varepsilon \\ R &\rightarrow \emptyset \\ R &\rightarrow R \cup R \\ R &\rightarrow RR \\ R &\rightarrow R^* \end{aligned}$$

I.e., a grammar is basically a set of *rewriting rules*, or *productions*. We can also abbreviate the grammar to:

$$R \rightarrow 0 \mid 1 \mid \varepsilon \mid \emptyset \mid R \cup R \mid R \circ R \mid R^*$$

Sentences

A simpler example is this grammar G:

$$\begin{aligned} A &\rightarrow \varepsilon \\ A &\rightarrow 0\ A\ 1\ 1 \end{aligned}$$

Using the two rules as a rewrite system, we get *derivations* such as

$$\begin{aligned} A &\Rightarrow 0A11 \\ &\Rightarrow 00A1111 \\ &\Rightarrow 000A111111 \\ &\Rightarrow 000111111 \end{aligned}$$

A is called a *variable* or *nontermina symbol*. Other symbols (here 0 and 1) are called *terminals* or *terminal symbols*.

The intermediate sequences that contain both variables and terminals are called *sentential forms*. The final sequence that contains only terminals is called a *sentence*.

Context-Free Languages

Clearly, each context-free grammar determines a language (a set of strings of terminals).

The language of grammar G (from the previous slide), denoted $L(G)$, is

$$L(G) = \{\, 0^n 1^{2n} \mid n \geq 0 \,\}$$

A language is called a *context-free language* (CFL) if it can be generated by some context-free grammar.

Some of the languages that we showed were not regular *are* context-free, for example

$$\{\, 0^n 1^n \mid n \geq 1 \,\}$$

The grammar for this language is simply

$$A \to 0A\,1 \mid 0\,1$$

Context-Free Grammars Formally

A context-free grammar (CFG) G is a 4-tuple (V, Σ, R, S), where

1. V is a finite set of *variables*,
2. Σ is a finite set of *terminals*,
3. R is a finite set of *rules*, each consisting of a variable (the left-hand side) and a sentential form (the right-hand side),
4. S is the *start variable*.

The binary relation $\Rightarrow$ on sentential forms is defined as follows.

Let u, v, and w be sentential forms. Then $uAw \Rightarrow uvw$ iff $A \to v$ is a rule in R.

I.e., $\Rightarrow$ captures a single derivation step.

Then $\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$, and

$$L(G) = \{s \in \Sigma^* \mid S \overset{*}{\Rightarrow} s\}$$

Examples

The following languages are context-free:

- $L = \{\, 0^m 1^n \mid m \leq n \,\}$
- $L = \{\, 0^m 1^m 2^n 3^n \mid m, n \geq 0 \,\}$
- $L = \{\, w \in \{0,1\}^* \mid w$ has an equal number of 0s and 1s $\,\}$
- $L = \{\, ww^R \mid w \in \{a,b\}^* \,\}$
- $L = \{\, w \in \{a,b\}^* \mid w = w^R \,\}$
- $L = \{\, w \in \{(,)\}^* \mid w$ is a balanced parenthesis string $\,\}$
- $L = \{\, s \in \{a,b\}^* \mid s \neq ww$, for any $w \,\}$
- Many programming languages.
- Simplified natural languages.

Regular languages are context-free

Theorem. Every regular language is context-free.

Proof. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA for a regular language L. Define a context-free grammar $G = (V, \Sigma, R, S)$ as follows:

- $V = Q$
- $R = \{\, p \to a\, q \mid \delta(p, a) = q \,\} \cup \{\, p \to \varepsilon \mid p \in F \,\}$
- $S = q_0$

Then, it is straightforward to show by induction on $|s|$ that G derives a string s if and only if A accepts s.

Derivations

Note 1. A CFL is regular iff it has a CFG in which every rule has the form $A \to \varepsilon$ or $A \to aB$, where A and B are variables and a is a terminal.

Note 2. More generally, a CFL is regular iff it has a CFG in which every rule has the form $A \to w$ or $A \to wB$, where A and B are variables and w is a sequence of terminals. This is sometimes called *right-linear normal form.*

Note 3. Every context-free language over $\Sigma = \{1\}$ is regular.

Exercise. Prove Note 3.

Derivations

A sequence of rewritings that transforms the start variable S of a grammar G to a sentence s is called a *derivation* of s from G.

A derivation in which every derivation step uses the leftmost variable in the sentential form is called a *leftmost* derivation.

A grammar G is called *ambiguous* if there exists a string s with two *different* leftmost derivations from G.

For example, the arithmetic expression grammar

$$E \to 0 \mid 1 \mid \ldots \mid 9 \mid (\, E \,) \mid E * E \mid E + E$$

is ambiguous because the sentence

$$2 + 3 * 4$$

has two different leftmost derivations.

Parse Trees

Here is another grammar for arithmetic expressions:

$$\begin{aligned} E &\rightarrow T \mid T + E \\ T &\rightarrow F \mid F * T \\ F &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \mid (\ E \ ) \end{aligned}$$

(When the start variable is unspecified, it is assumed to be the variable of the first rule, in this case E.)

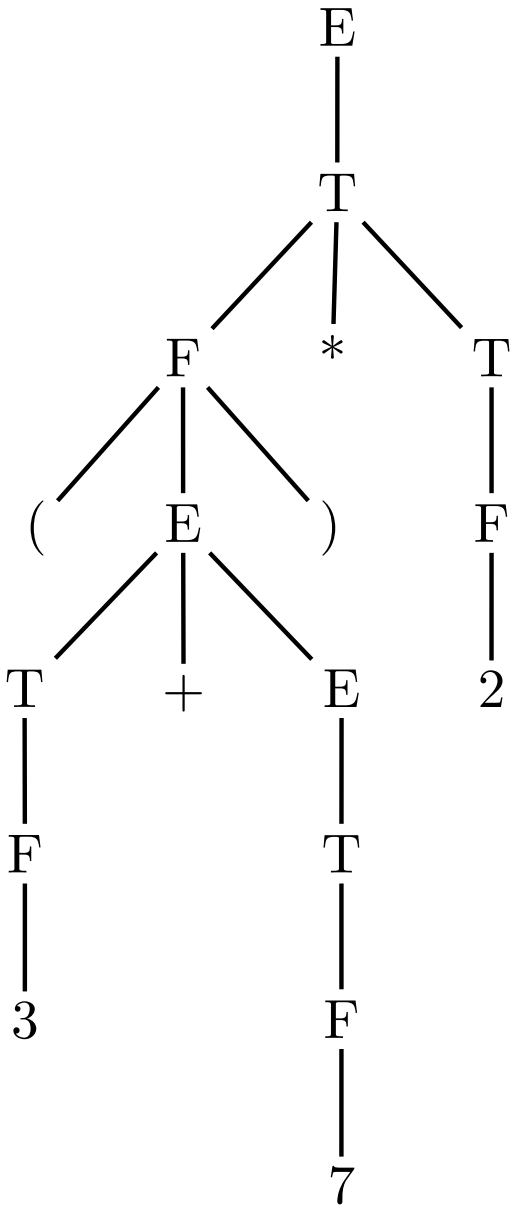
This grammar is unambiguous. (Convince yourself of this fact.)

Moreover, this grammar ensures that $*$ binds tighter than +.

So it is a "better" grammar than the previous one. (And it emphasises the fact that there may be multiple grammars for the same language.)

Parse Trees (cont.)

Here is a *parse tree* for `(3 + 7) * 2`:



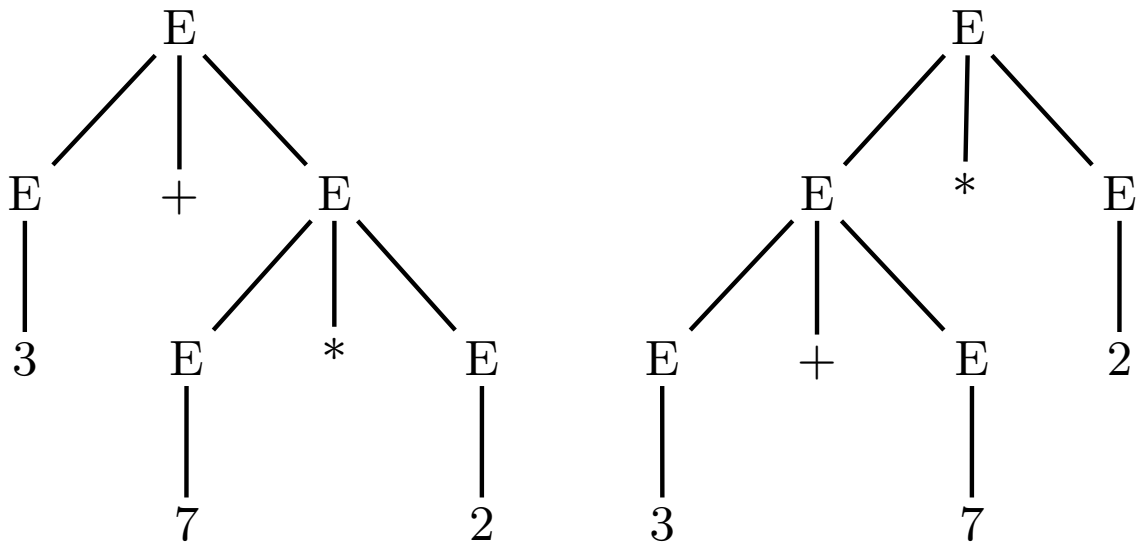
Parse Trees (cont.)

This is the *only* parse tree for this sentence (using this second grammar).

In contrast, consider the previous grammar

$$E \quad \rightarrow \quad 0 \mid 1 \mid \ldots \mid 9 \mid (\ E \ ) \mid E * E \mid E + E$$

This grammar has two *different* parse trees for the sentence $3 + 7 * 2$:



Ambiguity (cont.)

Previously, we said a grammar was ambiguous if there exists some sentence with two differentl leftmost derivations.

Equivalently, a grammar is ambiguous if there exists some sentence with two different parse tree.

Sometimes we can find a better grammar (as in our example) which is not ambiguous, and which generates the same language.

However, this is not always possible: There are CFLs that are *inherently ambiguous*, for example,

$$L = \{ a^i b^j c^k \mid i = j \text{ or } j = k \}.$$

(For any grammar for L, there are two different parse trees for $a^3 b^3 c^3$.)

Chomsky Normal Form

It is sometimes convenient to transform a CFG into a normal form.

A CFG is in *Chomsky normal form* (CNF) if every rule has one of the following forms:

$$\begin{aligned} S &\rightarrow \varepsilon \\ A &\rightarrow a \\ A &\rightarrow B\ C \end{aligned}$$

where S is the start variable, A *may* be the start variable, B and C are (non-start) variables, and a is a terminal.

Theorem. Every CFL has a CFG in CNF.

CNF Transformation

To transform an arbitrary CFG into CNF (S, Thorem 2.9):

1. Add a new start symbol S_0.
2. Eliminate all ε symbols not involving S_0.
3. Eliminate all unit rules $A \to B$.
4. Transform all remaining rules into the correct form.

Exercise. Construct a CNF grammar for the language of arithmetic expressions.

Griebach Normal Form

Another important normal form is *Griebach normal form* (GNF), in which every rule has one of the following forms:

$$\begin{aligned} S &\rightarrow \varepsilon \\ A &\rightarrow aB_1 \dots B_n \end{aligned}$$

where S is the start variable, A *may* be the start variable, $B_1, \dots, B_n$ are (non-start) variables, and a is a terminal.

Theorem. Every CFL has a CFG in Griebach normal form.

Exercise. Construct a GNF grammar for the language of arithmetic expressions.

Both these normal forms are important for different purposes.

Not every language is context-free

The following languages are *not* context-free:

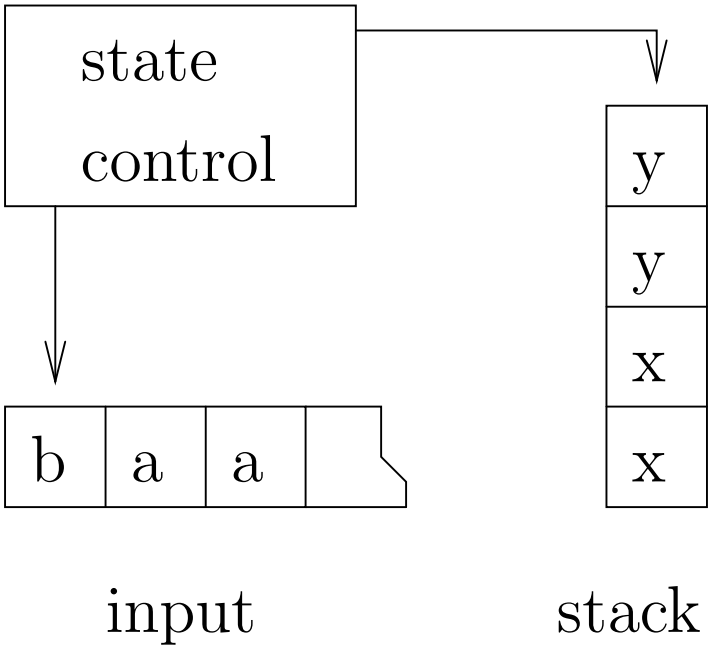
- $L = \{ 0^n 1^n 2^n \mid n \geq 0 \}$
- $L = \{ ww \mid w \in \{a, b\}^* \}$
- $L = \{ 0^{n^2} \mid n \geq 0 \}$
- The set of legal Java class definitions.
- The set of correct English sentences.

We describe later how to prove languages are *not* context-free...

Pushdown Automata

The automata we considered so far were limited by their *lack of memory*.

A *pushdown automaton* (PDA) is a nondeterministic, finite-state automaton, equipped with a *stack*.



The language $\{ a^i b^i \mid i \geq 0 \}$ is not recognised by any DFA as it requires the DFA to remember how many a's were in the input (and it can't do this).

Pushdown Automata (cont.)

(Initially), we consider *non-deterministic* PDAs.

A PDA may, in one transition step, read a symbol from input and read the top stack symbol.

Based on the current state, input symbol and stack top, it may change to a new state, pop the stack top, and push a sequence of symbols onto the stack.

It may ignore any input symbol (an ε-transition).

It may choose not to pop the stack (another ε-transition) and/or not to push anything onto the stack.

(Hmmm, seems a bit complicated...)

Pushdown Automata Formally

A pushdown automaton is a 6-tuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where

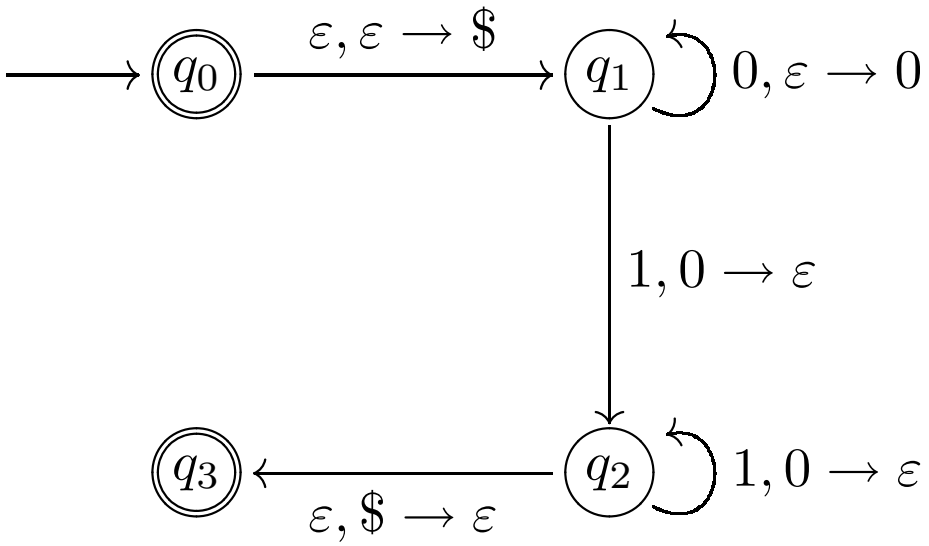
- Q is a finite set of *states*,
- Σ is a finite *input alphabet*,
- Γ is a finite *stack alphabet*,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma^*)$ is the *transition function*,
- $q_0 \in Q$ is a *start state*, and
- $F \subseteq Q$ are the *final states*.

Here, $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.

(This definition is more general than Sipser's, but it is *not* more expressive.)

PDA Example 1

This PDA recognises $\{0^n 1^n \mid n \geq 0\}$:



Acceptance Precisely

The PDA $(Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts input w iff $w = w_1 w_2 \cdots w_n$ with each $w_i \in \Sigma_\varepsilon$, and there are states $r_0, r_1, \ldots, r_n \in Q$ and strings $s_0, s_1, \ldots, s_n \in \Gamma^*$ such that

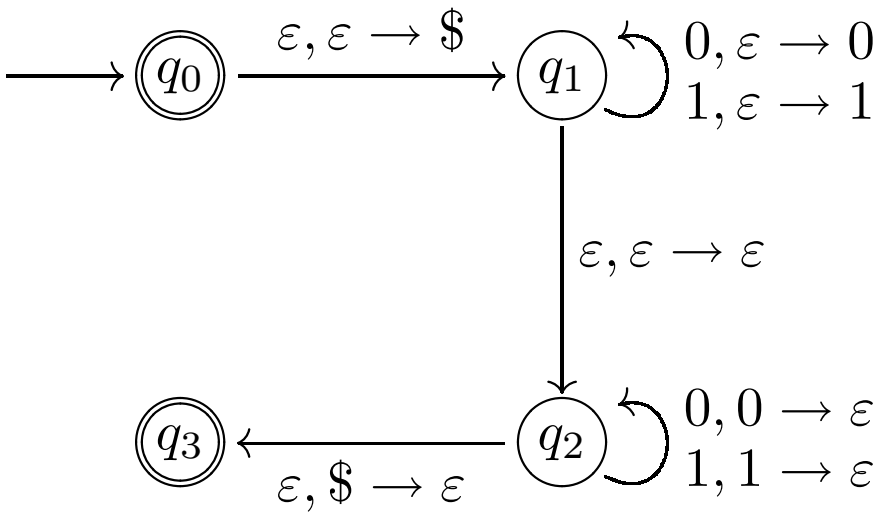
1. $r_0 = q_0$ and $s_0 = \varepsilon$.
2. $(r_{i+1}, b_1 \ldots b_k) \in \delta(r_i, w_{i+1}, a)$, $s_i = as$, $s_{i+1} = b_1 \ldots b_k s$, with $a \in \Gamma_\varepsilon$, $b_1 \ldots b_k \in \Gamma^*$ and $s \in \Gamma^*$.
3. $r_n \in F$.

Note 1 There is no requirement that $s_n = \varepsilon$, so the stack may be non-empty when the PDA halts (even if it accepts).

Note 2 Trying to pop an empty stack leads to nonacceptance of input, not to "runtime error".

PDA Example 2

This PDA recognises $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$:



Note that this PDA is (very) nondeterministic: at any time in state q_1, it can either continue to read more of w or it can move to state q_2 and start reading w^R.

More Examples

Exercise. Construct a PDA that recognises strings of a's and b's with an equal number of a's and b's.

Exercise. Construct a PDA that recognises

$$L = \{\, 0^i 1^j 2^k \mid i = j \text{ or } j = k \,\}$$

Hint. Choose nondeterministically whether to recognise strings with $i = j$ or with $j = k$.

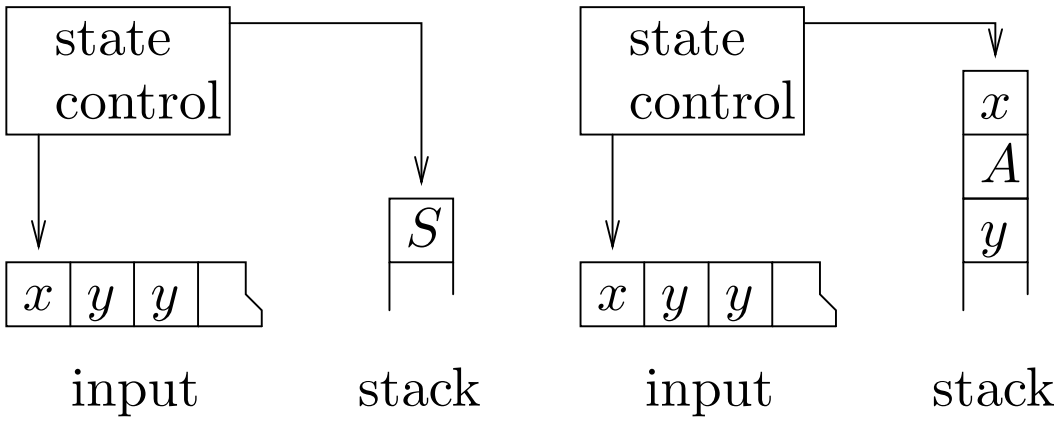
Exercise. Construct a PDA that recognises the set of arithmetic expressions constructed from an identifer a, operators $+$ and $\times$, and parentheses.

CFLs Have PDAs

Lemma. Every context-free language L is recognised by some PDA.

Proof. Given a CFG G, we construct a PDA P such that $L(P) = L(G)$. The PDA uses its stack to store a list of pending recogniser tasks.

For example, if $S \to xAy$ is a rule in G, then the PDA may replace an S on top of its stack by the sequence x, A, y.

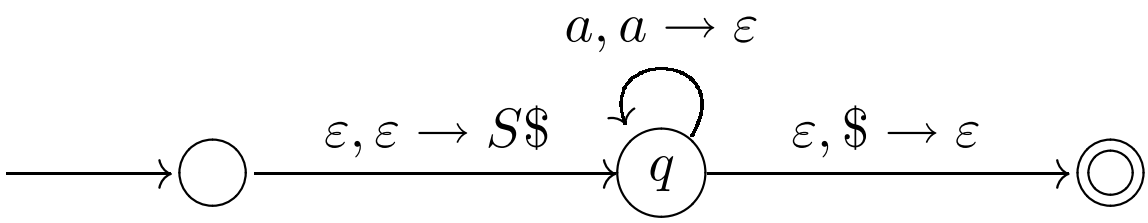


If x is the next input symbol and x is on top of the stack, then the PDA may consume x and pop the stack.

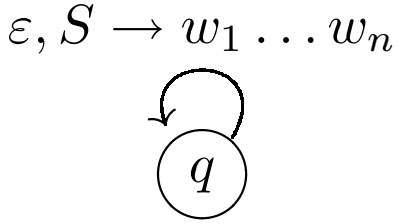
CFLs Have PDAs (cont.)

Construct the PDA with an initial state, an intermediate state q, and a final state.

Add a self-loop from q for each terminal a.

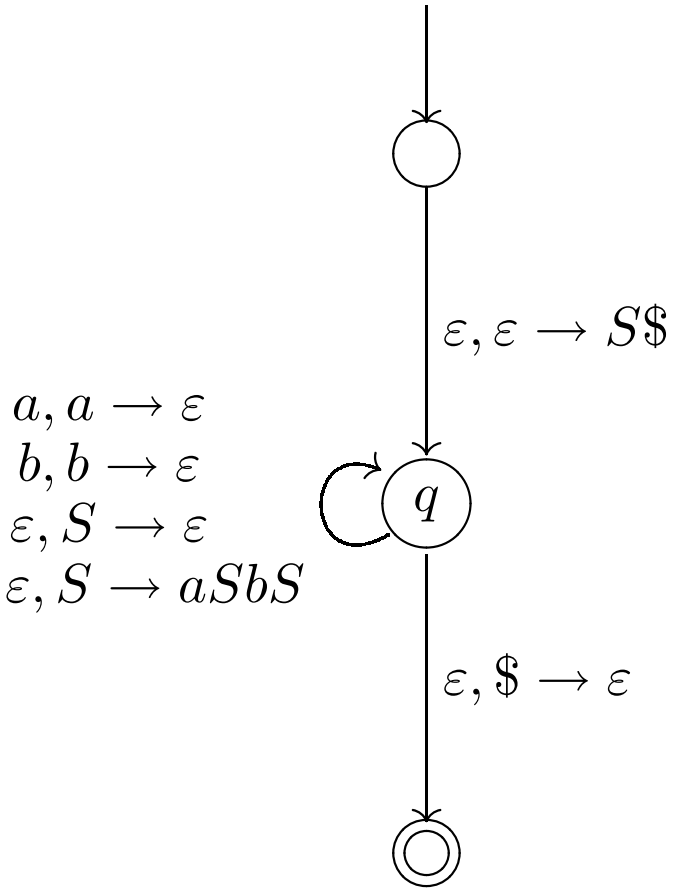


Also add, for each rule $A \to w_1 \ldots w_n$, another self-loop from q.



Example PDA

For the grammar $S \to \varepsilon \mid aSbS$ we get



PDAs Recognise CFLs

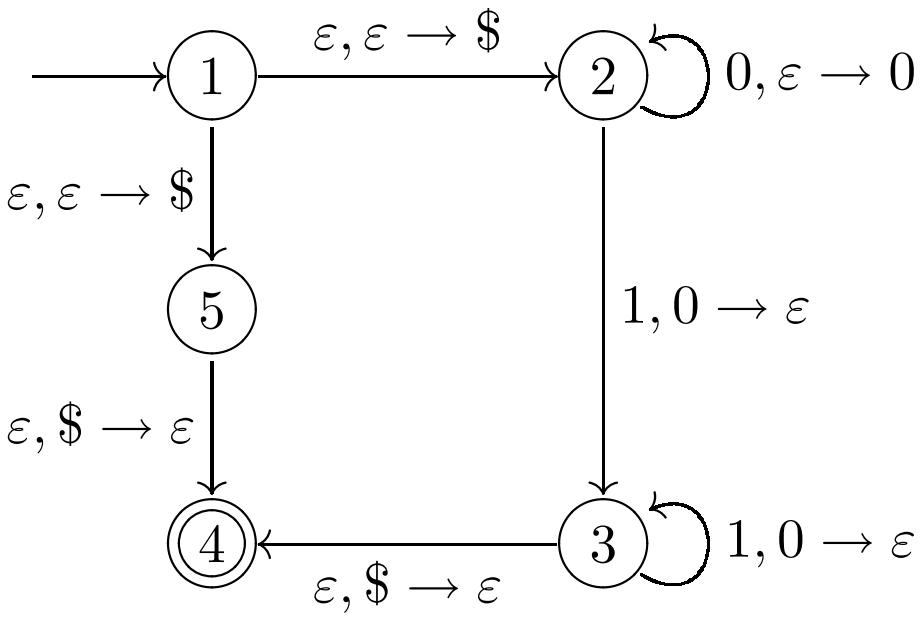
Lemma. Every language recognised by some PDA is context-free.

Proof outline. We show how to construct a CFG G which "simulates" the given PDA P.

Without loss of generality, we assume that P has only one accept state, q_f, that P empties its stack before accepting, and that each transition either pops or pushes a symbol (but not both).

The variables will be A_{pq} where p and q are states in P. Each A_{pq} will generate a string w iff w takes P from state p to state q leaving the stack unchanged. The start variable will be $A_{q_0 q_f}$ where q_0 is the start state of P.

Example



Identify all pairs of transitions where the same stack symbol is pushed then popped.

For \$: $A_{14} \to \varepsilon\ A_{23}\ \varepsilon \mid \varepsilon\ A_{55}\ \varepsilon \mid \varepsilon\ A_{25}\ \varepsilon \mid \varepsilon\ A_{53}\ \varepsilon$

For 0: $A_{23} \to 0\ A_{22}\ 1 \mid 0\ A_{23}\ 1$

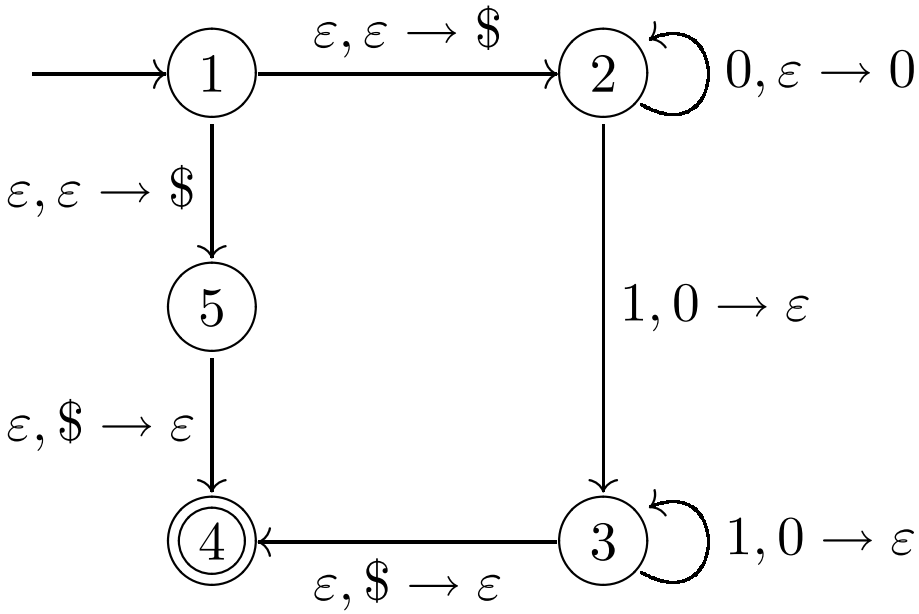
Add the five rules: $A_{ii} \to \varepsilon$

Add the 125 rules: $A_{ik} \to A_{ij}\ A_{jk}$

Fortunately, most of these variables are unreachable.

Example (cont.)

Cleaning up, the PDA



is simulated by the CFG

$$\begin{aligned} A_{14} &\to A_{23} \mid \varepsilon \\ A_{23} &\to 0\ 1 \mid 0\ A_{23}\ 1 \end{aligned}$$

PDAs Recognise CFLs Precisely

The construction precisely:

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_f\})$.

The variables of G are $\{A_{pq} \mid p, q \in Q\}$ and the start variable is $A_{q_0 q_f}$.

- Add rule $A_{pq} \to a\ A_{rs}\ b$ whenever $\delta(p, a, \varepsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ε).
- Add rule $A_{pq} \to A_{pr}\ A_{rq}$ for all $p, q, r \in Q$.
- Add rule $A_{pp} \to \varepsilon$ for all $p \in Q$.

We then have: A_{pq} generates x iff x can bring P from p to q with unchanged stack.

The detailed proof is by induction on the length of the derivation $A_{pq} \stackrel{*}{\Rightarrow} x$.

PDAs Recognise Exactly the CFLs

From these two lemmas:

Theorem. A language L is context-free if and only if it is recognised by some PDA.

Since every NFA is also a PDA (which ignores its stack), we have another proof of the fact that every regular language is context-free.

Note again that PDAs are nondeterministic! We describe later the properties of *deterministic* PDAs...

First, we show how to prove languages are *not* context-free.

Pumping Lemma for CFLs

If A is context-free, there exists a number $p \geq 0$ such that every string $s \in A$ with $|s| \geq p$ can be written as $s = uvxyz$, where

1. $|vxy| \leq p$
2. $|vy| > 0$
3. $uv^i xy^i z \in A$ for all $i \geq 0$

(Wow!)

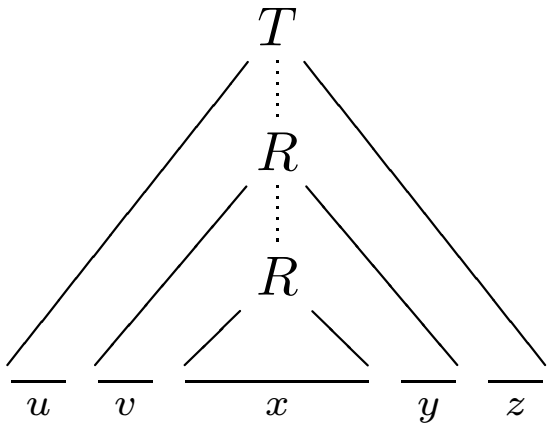
Proving the Pumping Lemma

Let T be the start variable of a CFG G which generates A.

Let b be the length of the longest right-hand side in G.

Set $p = b^{|V|+2}$.

Consider some string s derived from T. If $|s| \geq p$ then the height of the parse tree is at least $|V| + 2$, as the tree has branching factor b or less.

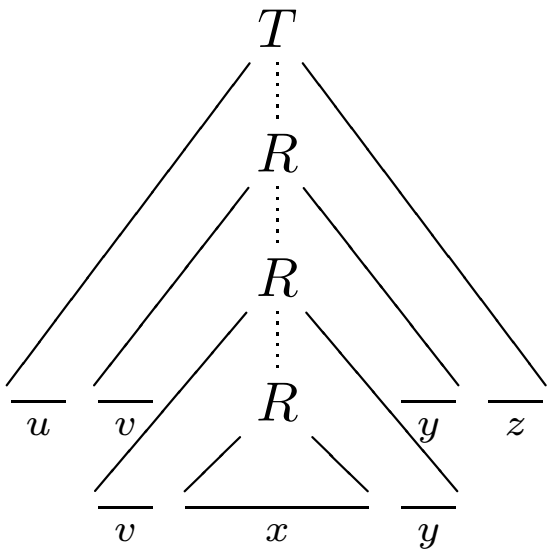


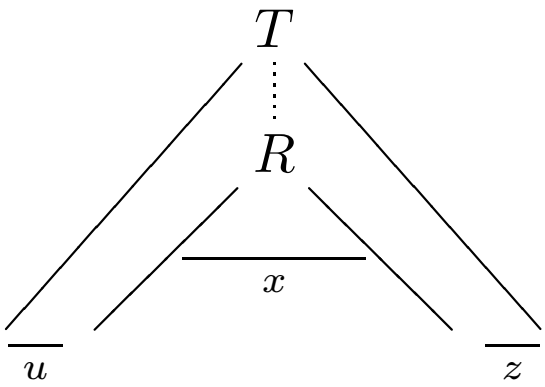
Hence the longest path has $|V| + 1$ variables or more, so some variable (*e.g.*, R) occurs repeatedly.

Proving the Pumping Lemma (cont.)

This gives the desired splitting into $uvxyz$.

Clearly these are also valid parse trees:





Proving the Pumping Lemma (cont.)

The condition that $|vxy| \leq p$ is satisfied if we make sure that the occurrences of R we consider are in the *lowest* part of the tree.

If both occurrences of R fall within the bottom $|V| + 1$ variables on the longest path, then the tree that generates vxy has height at most $|V| + 2$.

And so it generates a string of length at most $b^{|V|+2}$, that is, p.

Example 1

$B = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is not context-free.

Assume it is, and let p be the pumping length.

Consider $a^p b^p c^p \in B$ with length greater than p.

By the pumping lemma, $a^p b^p c^p = uvxyz$, with $uv^i xy^i z$ in B for all $i \geq 0$.

Either v or y is non-empty.

If one of them contains two different symbols from $\{a, b, c\}$ then uv^2xy^2z has symbols in the wrong order, and so cannot be in B.

So both v and y must contain only one kind of symbol. But then uv^2xy^2z can't have the same number of as, bs, and cs (because we've increased the number of only one or two of them).

In all cases we have a contradiction.

Example 2

$D = \{ww \mid w \in \{0,1\}^*\}$ is not context-free.

Assume it is, and let p be the pumping length.

Consider $0^p1^p0^p1^p \in D$.

By the pumping lemma, $0^p1^p0^p1^p = uvxyz$, with uv^ixy^iz in D for all $i \geq 0$, and $|vxy| \leq p$.

There are three ways that vxy can be part of

$$00\ldots0011\ldots1100\ldots0011\ldots11$$

If it straddles the midpoint, it has form 1^n0^m, so pumping down, we are left with $0^p1^i0^j1^p$, with $i < p$, or $j < p$, or both.

If it is in the first half, uv^2xy^2z will have pushed a 1 into the first position of the second half.

Similarly if vxy is in the second half.

In all cases the result is not in D.

Closure Properties for CFLs

The class of context-free languages is closed under

- union,
- concatenation,
- repetition (Kleene star),
- reversal.

They are not closed under intersection!

Consider these two CFLs:

$$\begin{aligned} E &= \{a^m b^n c^n \mid m, n \in \mathbb{N}\} \\ F &= \{a^n b^n c^m \mid m, n \in \mathbb{N}\} \end{aligned}$$

Exercise. Prove that they are context-free!

But $E \cap F$ is the language $B = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ which we just proved not to be context-free.

However, we do have: If A is context-free and R is regular then $A \cap R$ is context-free.

Regular Grammars

We have seen "context-free grammars".

Is there a grammar formalism that corresponds to the regular languages?

If we restrict the kind of rules allowed in CFGs, so they must be either of form

$$A \to w$$

or

$$A \to w\ B$$

with $w \in \Sigma^*$ and $A, B \in V$, then we have "regular grammars".

These generate exactly the regular languages.

(Here we chose the *right-linear* form — we could have said $A \to B\ w$ instead of $A \to w\ B$.)

Deterministic PDAs

A *deterministic PDA* (DPDA) is a PDA for which there is at most one possible transition for every (state, input symbol, stack symbol)-triple.

Exercise. Is the PDA in Example 1 deterministic or nondeterministic? If it is nondeterministic, construct an equivalent DPDA.

Exercise. Construct a DPDA for the language of regular expressions.

Theorem. Not every context-free language can be recognised by a deterministic PDA.

Thus, nondeterminism adds real expressive power to pushdown automata!

Also, every DPDA-recognisable language has an unambiguous grammar.

Deterministic PDAs (cont.)

Example. A DPDA can recognise the context-free language

$$\{wcw^{\mathcal{R}} \mid c \in \Sigma, w \in (\Sigma \setminus \{c\})^*\}$$

but not the context-free language

$$\{\, ww^{\mathcal{R}} \mid w \in \Sigma^* \,\}.$$

The intuition is that a deterministic PDA cannot know when the middle of the input has been reached. *E.g.*, suppose it reads

`00001100000000110000`

How can the deterministic PDA know when to start popping the stack?

On the other hand, efficient parsing must be done deterministically, and hence must be restricted to CFLs recognised by DPDAs.

Decision Problems for CFLs

The following problems are decidable:

1. *Emptiness.* Is CFL L empty?
2. *Finiteness.* Is CFL L finite?
3. *Membership.* Does string w belong to CFL L?

The following problems are undecidable!

1. *Ambiguity.* Is CFG G ambiguous?
2. *Inherent ambiguity.* Is CFL L inherently ambiguous?
3. *Empty intersection.* Is the intersection of two CFLs L_1 and L_2 empty?
4. *Equality.* Are two CFLs L_1 and L_2 equal?
5. *Totality.* Is the CFL L equal to Σ^*?