COSC 341
Theory of Computing
Lectures 9 and 10
Pushdown automata and context-free grammars

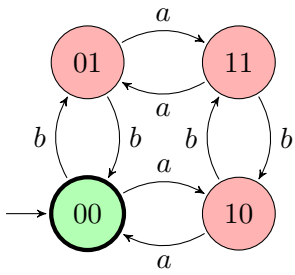
Stephen Cranefield
stephen.cranefield@otago.ac.nz

Lecture slides (mostly) by Michael Albert

Keywords: Pushdown automata, context-free grammars, context-free languages, pumping lemmas

Memory

- The states of a DFA (or NFA) can be thought of as its memory
- For instance in the EVEN-EVEN machine below (Tut. 4, Q3) we used four states representing two bits – the parity of a's and of b's.
- The capacity of this memory is limited and fixed
- If we wanted to recognise a language like $\{a^n b^n : n \geqslant 0\}$ we'd need some form of unbounded memory.



Pushdown automata

A pushdown automaton (PDA) is, fundamentally, a finite state machine augmented with a stack of unbounded capacity. That entails some required changes to the description of how transitions work.

Q	a finite set of states,
Σ	the input alphabet (lower case letters)
Γ	the stack alphabet (upper case letters)
δ	a non-deterministic transition function
q_0	the initial state
F	the set of accepting states

What does the δ do?

The inputs to the transition function, δ are:

- The current state
- An input letter (from Σ) or ϵ (meaning no letter is consumed)
- The stack top letter (from Γ) or ϵ (meaning the stack is not popped)

Its output is a set of possible transitions each of which consists of:

- A new state
- A new stack top or ϵ (meaning nothing is pushed)

The intended action is:

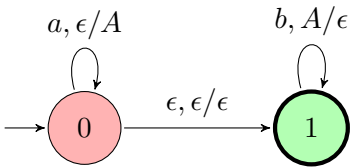
- Consume the input letter (if any)
- Pop the stack top letter (if any)
- Push the new stack top (if any)
- Move to the new state

When do we accept?

All of the following must be true:

- No more input
- Stack is empty
- State is accepting

Accepting $\{a^n b^n \ : \ n \geqslant 0\}$



The transition notation on the arrows is

<input letter>, <stack top> / <new stack top>

Remember that if a stack top is specified it is popped.

Can we peek?

What if we want to do the following?

On input a if the stack top is A, then pop it (and don't push anything) but if it's B then push an A (without popping).

The first part is standard, the second requires a "peek" – but we can emulate it as follows by introducing some new states:

- Define an $(a, B/B)$ transition to a new state,
- Whose only transition is $(\epsilon, \epsilon/A)$ (to whatever state you want to be in after the full transition).

Similarly we can emulate "peeking to a bounded depth" or "multiple pushes in one transition"

Context-free grammars (CFGs)

A context-free grammar is a grammar whose productions are of the form:

<non-terminal> $\longrightarrow$ <any word in terminals and non-terminals>

For instance:

$$S \longrightarrow \epsilon \,|\, aSb$$

The language it generates is the set of all strings without non-terminal letters that can be produced – in the case above $\{a^n b^n \,:\, n \geqslant 0\}$.

A theorem

Theorem

The sets of languages that can be accepted by pushdown automata and generated by context-free grammars are the same.

That a CFG can be accepted by a PDA is not too difficult

- Start with a transition that pushes the start non-terminal onto the stack.
- If the stack-top is non-terminal push one of its productions, a letter at a time, onto the stack (so that leftmost winds up on top)
- If the stack-top is terminal allow a transition that consumes the corresponding symbol from input (and pops the stack-top)

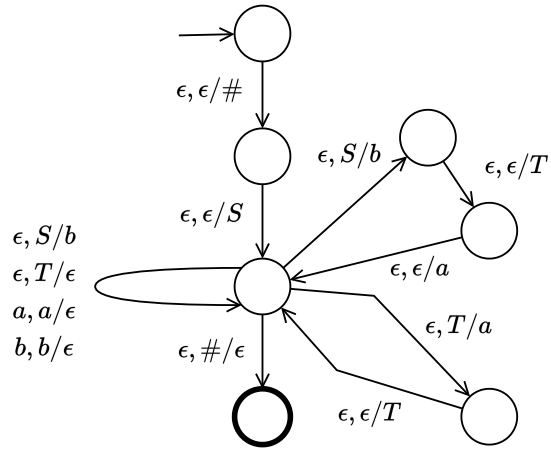
The other direction is ugly and non-informative.

These languages are known as context-free languages.

Example CFG to PDA translation

$S \to b \mid aTb$

$T \to \epsilon \mid Ta$



- Based on M. Sipser, Introduction to the theory of computation, Fig. 2.12.
- '#' is used to mark the bottom of the stack.

Closure properties for context-free languages

The class of context-free languages is closed under:

- Union
- Concatenation
- Kleene-star
- Intersection with regular languages

Notably missing above are intersection (between context-free languages) and complement.

Example of intersection between context-free languages:

$L_1 = \{a^n b^n c^t : n, t \geq 0\}$ is context-free (easy extension of PDA for $\{a^n b^n\}$ above)

$L_2 = \{a^s b^n c^n : n, t \geq 0\}$ is context-free (easy modification of PDA for L_1)

$L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0\}$ is not context-free (shown in a later slide)

Pumping lemma for regular languages

Theorem

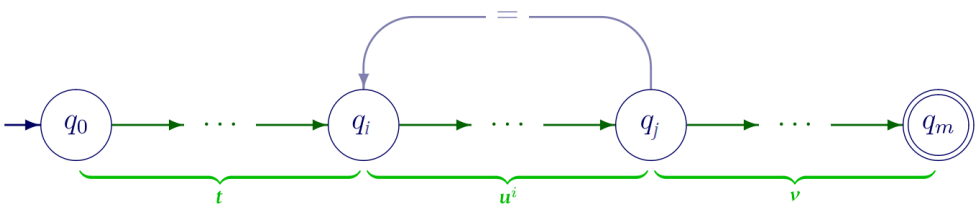
If a language, L is regular, then there exists a positive integer k such that, for any $w \in L$ with $|w| \geqslant k$ there exist $t, u, v \in \Sigma^*$ such that:

- $w = tuv$,
- $|u| > 0$,
- $|tu| \leq k$, *and*
- *for all* $i \geqslant 0$, $tu^i v \in L$.

That is, for sufficiently long words in L we can "pump" some short internal segment an arbitrary number of times.

Proof idea: If L is regular then there is a DFA that accepts it. Take k to be greater than the number of states in the DFA. Any accepted word, w, of length at least k must revisit some state. Take u to be the part of w that is consumed between visits.

Proof idea illustrated



Orignal diagram by Jochen Burghardt, CC BY-SA 4.0, via Wikimedia Commons. Modified by Stephen Cranefield to change variable names.

Pumping lemma for context-free languages

Theorem

If a language, L is context-free, then there exists a positive integer k such that, for any $w \in L$ with $|w| \geqslant k$ there exist $r, s, t, u, v \in \Sigma^*$ such that:

- $w = rstuv$,
- $|s| + |u| > 0$,
- $|stu| \leq k$, *and*
- *for all* $i \geqslant 0$, $rs^i tu^i v \in L$.

Proof idea: Any sufficiently long word in the language has a deep derivation tree. Any long enough branch contains a repeated non-terminal. Replace the derivation on the second instance by that on the first (to pump up) or vice-versa (to pump down).

Proof idea illustrated

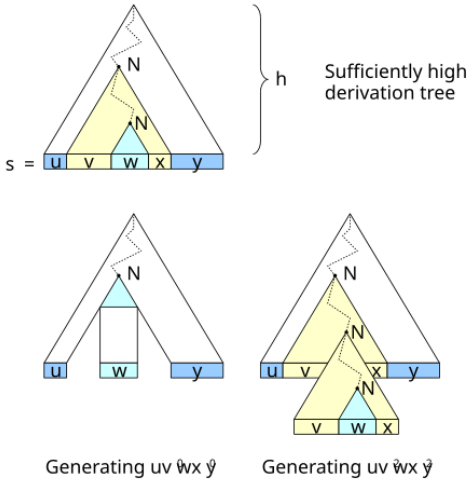


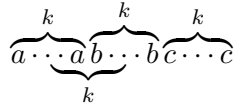
Diagram by Jochen Burghardt, Licence: CC BY-SA 3.0.

Use of the pumping lemmas

- The pumping lemmas present a one-way inference
  (L is regular/context-free $\rightarrow$ some property on words)
- The reverse implication does not hold
- They can be used to prove that a language is not regular or not context-free:
  - Assume the language is in that class
  - Then the pumping lemma property would hold
  - Derive a contradiction

Example: $L = \{a^n b^n c^n : n \geq 0\}$ is not context-free

- If it is context free, the pumping lemma would apply for some constant k.
- Consider the word $w = a^k b^k c^k$, which is in L.
- Try to slide a window of length k along the word w



  The window cannot contain more than two of the letters a, b and c
- Therefore, for any factorisation of w into $rstuv$ with $|stu| \leq k$ one letter is missing from stu
- If we pump s and u in w to get $rsstuuv$, we will still only have k copies of one of the letters, but $k + 1$ copies of the letter(s) in r and u.
- Therefore $rs^2tu^2v \notin L$
- That contradicts the pumping lemma, so L cannot be context-free.

The model matters

A pushdown automaton cannot accept the language:

$$\textsc{Powers-of-2} = \{a^n \,:\, n = 2^k \text{ for some } k \geqslant 0\}$$

Why?

According to the pumping lemma, if we could, then for any "large enough" power of two, n, and some "small" t, we would have to accept a^{n+jt} for any $j \geqslant 0$. But the gaps between consecutive powers of two get bigger and bigger so this is not possible.

What if we use a queue instead of a stack?

A queue machine (or pullup automaton, PUA) for POWERS-OF-2

Set-up: Add a $\#$ to the queue. Then on each input a, add an A to the queue. Make a non-deterministic transition to the checking phase.

Checking phase: The queue encodes a number x_i as

$$\#\overbrace{A\cdots A}^{x_i} \quad \text{(the queue head is on the left)}$$

Idea: Reject if $x_i = 0$. Otherwise, compute $x_i \operatorname{rem} 2$ (left of $\#$) and $x_i \operatorname{div} 2$ (right of $\#$). As long as $x_i \operatorname{rem} 2 = 0$ and $x_i \operatorname{div} 2 \neq 1$, set $x_{i+1} = x_i \operatorname{div} 2$ and recurse. Accept when $x_i \operatorname{div} 2 = 2^0 = 1$ and $x_i \operatorname{rem} 2 = 0$.

1. If queue head is $\#$, move it from head to tail and:
   - 1.1 If head is $\#$ again, there were no As so reject ($x_i = 0$).
   - 1.2 Remove an A.
     - 1.2.1 If head is $\#$, accept ($x_i = 1$)
     - 1.2.2 Else go to 2.2.
2. Remove an A.
   - 2.1 If queue head is $\#$, reject ($x_i \operatorname{rem} 2 = 1$).
   - 2.2 Remove a second A (if present at head), add it to the tail and go to 1.

What languages can queue machines accept?

The POWERS-OF-2 implementation shows us that queue machines can accept more than context-free languages. In fact, they are equivalent to Turing machines (our next topic), i.e. queue machines and Turing machines can simulate each other.

However, queue machines may be less efficient than pushdown automata when recognising some context-free languages, e.g. recognising palindromes.