

Pushdown Automata

Robert M. Keller
Harvey Mudd College
1 October 2003

Wanted:

- Similar to the DFA characterization of type 3 languages, we'd like a **machine characterization** of type 2 languages.
- We know that finite-state machines are inadequate.
- We need to add an extra memory component, one that permits **unbounded storage**.

A Proper Context-Free Language

- Example: $\{xcx^R \mid x \in \Sigma^*\}$ where $c \in \Sigma$.
- Supposing $\Sigma = \{a, b\}$, give a context-free grammar for this language.
- Is this language regular?

Stacks to the Rescue

- By adding a stack to a FSA, we can accept non-regular languages.
- Example: $\{xcx^R \mid x \in \Sigma^*\}$ where $c \in \Sigma$.
 - Begin reading symbols.
 - Until we encounter a 'c', **push** the symbols read onto a stack.
 - After 'c' is encountered, **pop** the symbols from the stack, comparing with the next symbol read, if any.
 - Accept when the stack is **empty**.
 - Have to check for no c's, more than one c, etc. and reject these cases.

PDA's

- The type of behavior described on the preceding slide is that of a (deterministic) PDA (DPDA).
- In general, PDA's are allowed to be **non-deterministic**.
- The situation for PDA's is different from FSA's: there is **no subset construction**.

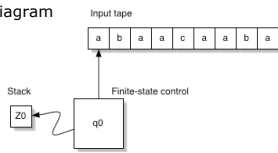
PDA Defined: $(Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$

- Q is a finite set of **control states**
- Σ is the **input** alphabet
- Γ is the **stack** alphabet
- q_0 is the **initial** control state
- Z_0 is the **initial** stack symbol
- $A \subseteq Q$ is the set of **accepting** control states
- δ is the **state transition relation** (or, in the case of a DPDA, **function**)

$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$

Say What?

PDA Diagram



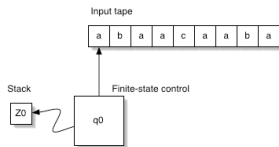
The **state** is generally of the form $(q, \Gamma \times \Sigma^* \times \Gamma^*)$, i.e. (control-state, stack-contents). Sometimes this is called a **configuration** or **instantaneous description (ID)**, to distinguish it from the control state. I prefer to call it the state, which is what it is.

The initial state is (q_0, Z_0) . We will describe Γ presently.

PDA Acceptance

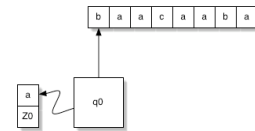
- There are different models for acceptance.
- **Final-state acceptance** means that the PDA accepts when it is in a **designated control state** after the input has been read.
- **Empty-stack acceptance** means that the PDA accepts when its **stack is empty** after the input has been read.
- The choice is a matter of convenience; the two can be shown **inter-convertible**.

Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

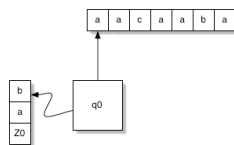


Note that control-state is in q_0 and will stay there until c is read.

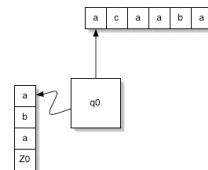
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



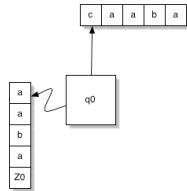
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



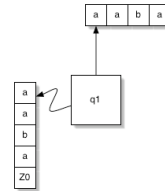
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

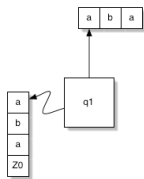


Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

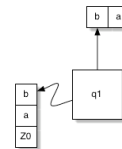


Note that control-state has changed to q_1 after reading c.

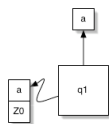
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



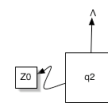
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



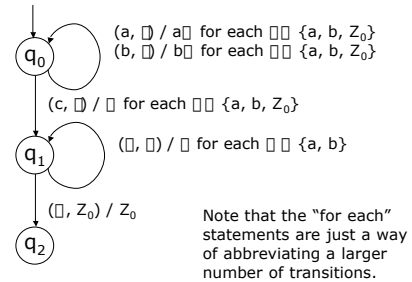
q_2 is the accepting control state. The input is now empty.

So the PDA **accepts** the original input: abaacaaba.

Design of the PDA for the xcx^R language

- Q : $Q \times (\Sigma \cup \{\epsilon\}) \times \Sigma$ finite subsets of $Q \times \Sigma^*$
 - $\{(q_0, \epsilon, \epsilon) = \{(q_0, \epsilon, \epsilon)\}$, for each $\{a, b\}, \epsilon \in \{a, b, Z_0\}$
 - $\{(q_0, c, \epsilon) = \{(q_1, \epsilon)\}$, for each $\{a, b, Z_0\}$
 - $\{(q_1, \epsilon, \epsilon) = \{(q_1, \epsilon)\}$, for each $\{a, b\}$
 - $\{(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$
- The top of the stack is the leftmost symbol in the string.
- The "for each" statements are just a way of abbreviating a larger number of separate transitions.
- For all other combinations, the RHS is empty, meaning that there is no transition defined. Unless the control state is accepting in this case, the input is **rejected** at this point.

δ can also be described as a graph



δ can also be described by "transition rules"

- $q_0, a, \epsilon \rightarrow q_0, a$ for each $\{a, b, Z_0\}$
- $q_0, b, \epsilon \rightarrow q_0, b$ for each $\{a, b, Z_0\}$
- $q_0, c, \epsilon \rightarrow q_1, \epsilon$ for each $\{a, b, Z_0\}$
- $q_1, \epsilon, \epsilon \rightarrow q_1, \epsilon$ for each $\{a, b, Z_0\}$, for each $\{a, b\}$
- $q_1, \epsilon, Z_0 \rightarrow q_2, Z_0$

The rules with "for each" are actually abbreviations for other rules with the ϵ and ϵ variables by literal symbols.

The important thing is that **the total number of rules is finite.**

The δ (turnstile) and δ^* notation.

- Let $q, q' \in Q$; $x, x' \in \Sigma^*$; $\epsilon, \epsilon \in \Sigma^*$
 - $(q, x, \epsilon) \vdash (q', x', \epsilon)$ means that there is a 1-step transition of the PDA from "state" (q, x, ϵ) to (q', x', ϵ) .
- \vdash^* is the **transitive closure** of \vdash , meaning:
 - $(q, x, \epsilon) \vdash^* (q, x, \epsilon)$
 - If $(q, x, \epsilon) \vdash^* (q', x', \epsilon)$ and $(q', x', \epsilon) \vdash^* (q'', x'', \epsilon)$, then $(q, x, \epsilon) \vdash^* (q'', x'', \epsilon)$.

The key property ("stack property") of PDA's:

- If $(q, x, \epsilon) \vdash^* (q', x', \epsilon)$ where $x, x' \in \Sigma^*$; $\epsilon, \epsilon \in \Sigma^*$ then for any $y \in \Sigma^*$ and $\epsilon \in \Sigma^*$ also $(q, xy, \epsilon) \vdash^* (q', x'y, \epsilon)$.
- In other words, steps that can take place with a given input and stack contents can also take place with additional input and lower stack contents, without depending upon or using the additional input or contents.

Use of Non-Determinism

- PDA's are non-deterministic in the general case: Some of the range values of δ can have more than one element.
- Unlike the situation with NFA's, this non-determinism is more of a mathematical artifice.
- It can also be used to understand various parsing algorithms.

Guessing Metaphor

- Think back to NFA's. One way to describe what they do in a given state with a given input is to "guess" which of several possible next states will ultimately be the "right" one (take the machine to an accepting state).
- This metaphor is extra helpful for PDA's.

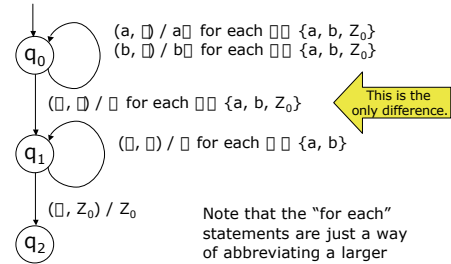
PDA accepting $\{xx^R \mid x \in \{a, b\}^*\}$

- While it appears similar to $\{xcx^R \mid x \in \{a, b\}^*\}$, this set is, in some sense, more difficult to recognize.
- The reason is that for a given input, we don't have an indicator of when x ends and x^R starts.
- Here a PDA can use its non-determinism: It can **guess** at the point it thinks x has ended.
 - If it guessed right, it will get to an accepting state.
 - If it guessed wrong, nothing lost.
 - An important thing is that guessing never leads to an accepting state when the input is not in the language.

PDA accepting $\{xx^R \mid x \in \{a, b\}^*\}$

- Here's how a PDA would work for this language:
 - It begins reading the input symbols.
 - For each symbol read, it would push the symbol onto its stack,
 - until it thinks it has reached the end of the x part of the input,
 - at which time it would transition to a new control state.
 - In the new control state, it would read input symbols, and pop the top of the stack, continuing as long as the symbol read matched the symbol on the stack.
 - Eventually it can guess has seen all of the input, and go to the accepting state, provided the top of the stack is Z_0 .

□ for $\{xx^R \mid x \in \{a, b\}^*\}$



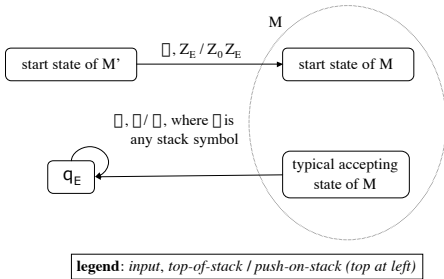
Interconvertibility of Acceptance Modes

- For every PDA M accepting by final state, there is a PDA M' accepting the same language by empty stack.** Proof:
 - Create M' from M as follows: Add a new state q_E and a transition from each accepting state of M to q_E .
 - Add transitions from q_E to itself which do nothing but pop symbols from the stack. This ensures that M' can empty its stack whenever M would have accepted.
 - However**, we must also ensure that M' empties its stack **only** in this case; M could have emptied its stack, so M' might do the same.

(Proving: For every PDA M accepting by accepting state, there is a PDA M' accepting the same language by empty stack.)

- Make the initial stack symbol of M' a new symbol Z_E not used in M .
- Transitions of M can never remove Z_E .
- Where M might have emptied its stack, M' will now have Z_E on the stack.
- By design, M' cannot remove Z_E unless doing so from q_E .

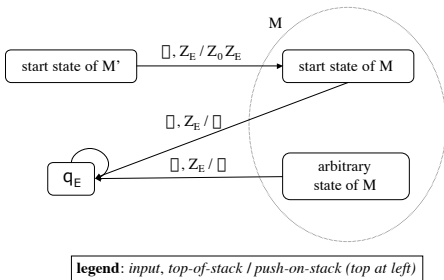
Accepting State \square Empty Stack



Interconvertibility of Acceptance Modes (2)

- For every PDA M accepting by empty stack, there is a PDA M' accepting the same language by accepting state. Proof:
- Create M' from M as follows: The empty stack symbol Z_E of M' is a new stack symbol not used in M .
- M' begins from a new initial state q_I , with a \square transition to the initial state of M , with which M' pushes $Z_0 Z_E$ where Z_0 is the initial stack symbol of M .
- Introduce a new accepting state q_F as the only accepting state of M' .
- Add new \square transitions from each state of M to q_F conditioned on being Z_E on top of the stack.
- Whenever M would have emptied its stack, M' will see Z_E and can make a transition to the accepting state.

Empty Stack \square Accepting State

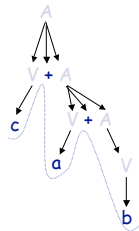


Leftmost/Rightmost Derivation Nomenclature

- A derivation in a context-free grammar is called leftmost if it is always the leftmost auxiliary that is replaced.
- A derivation in a context-free grammar is called rightmost if it is always the rightmost auxiliary that is replaced.
- Observation: For each derivation tree there is exactly one leftmost and one rightmost derivation.

Left/Rightmost Derivations

$A \square V \mid V+A$
 $V \square a \mid b \mid c$



leftmost: $A \square V+A \square c+A \square c+V+A \square c+a+A \square c+a+V \square c+a+b$

rightmost: $A \square V+A \square V+V+A \square V+V+V \square V+V+b \square V+a+b \square c+a+b$

CFL Characterization Theorem

- A language is context-free iff there is a pushdown acceptor that recognizes it.

Parallel Between CFG and PDA

- Each derivation $S \Rightarrow^* x$ in a CFG corresponds to a series of moves $(q, x, S) \vdash^* (q', \square, \square)$ of some PDA accepting by empty stack.

CFL \iff PDA Lemma

- Every context free language is accepted by some PDA.

1st Proof of the CFL \iff PDA Lemma: **top-down** or **produce-match** technique

- Assume L is a context-free language. Then L has a context-free grammar G in Greibach normal form.
- (It is easy to dispose of the Greibach normal form assumption, but we keep it initially for simplicity).
- Construct a PDA M with one control state q .
- This machine will accept by **empty-stack**.
- Each production, which has the form: $A \Rightarrow B_1 B_2 B_3 \dots B_n$, where A is terminal and each B_i is auxiliary, add to M the transition: $(q, \square, A) \vdash (q, B_1 B_2 B_3 \dots B_n)$
- The initial stack symbol is the start symbol S of G .
- We claim that for any $x \in \Sigma^*$ $(q, x, S) \vdash^* (q, \square, \square)$ iff $S \Rightarrow^* x$. This requires an inductive proof (given later).

Example of CFG to Machine

- | Production | Transition |
|----------------------|-------------------------------|
| $S \Rightarrow (T$ | $q, '(, S \vdash q, T$ |
| $S \Rightarrow (ST$ | $q, '(, S \vdash q, ST$ |
| $S \Rightarrow (TS$ | $q, '(, S \vdash q, TS$ |
| $S \Rightarrow (STS$ | $q, '(, S \vdash q, STS$ |
| $T \Rightarrow)$ | $q, ')', T \vdash q, \square$ |

Example accepting sequence

- Input: $((()))$
- Rules are:
 - $q, '(, S \vdash q, T$
 - $q, '(, S \vdash q, ST$
 - $q, '(, S \vdash q, TS$
 - $q, '(, S \vdash q, STS$
 - $q, ')', T \vdash q, \square$
- State sequence:
 - $q, ((())), S$ use rule: $q, '(, S \vdash q, ST$ to get
 - $q, (())$, ST use rule: $q, '(, S \vdash q, TS$ to get
 - $q, ())$, TST use rule: $q, ')', T \vdash q, \square$ to get
 - $q, ()$, ST use rule: $q, '(, S \vdash q, T$ to get
 - $q,)$, TT use rule: $q, ')', T \vdash q, \square$ to get
 - $q,)$, T use rule: $q, ')', T \vdash q, \square$ to get
 - q, \square, \square accept by empty stack

What's going on here?

- The pda is simulating a **leftmost derivation** of a string.
- In a leftmost derivation, the leftmost auxiliary is always rewritten.
 - The symbols to the left of that auxiliary in the derived string are the ones that have been read.
 - The symbols to the right are the stack contents.

Simulating Leftmost Derivation

- $S \rightarrow (T \quad q, (, S \rightarrow q, T$
 $S \rightarrow (ST \quad q, (, S \rightarrow q, ST$
 $S \rightarrow (TS \quad q, (, S \rightarrow q, TS$
 $S \rightarrow (STS \quad q, (, S \rightarrow q, STS$
 $T \rightarrow) \quad q,), T \rightarrow q, \emptyset$
- $((())) \mid S$ Red shows matched portions of input not remaining in input and not actually on the stack.
- $((())) \mid (ST$
- $((())) \mid ((ST$
- $((())) \mid ((ST$ Underscore shows the LHS symbol being rewritten.
- $((())) \mid ((())I$
- $((())) \mid ((())$

Exercise

- Show that Greibach normal form is not essential for the lemma and proof technique.
- (Allow terminals as well as auxiliaries on the stack. Allow ϵ to be either ϵ or an element of Σ .)

PDA \iff CFL Lemma

- For every PDA M , there is a context-free grammar G such that $L(G) = L(M)$.
- Proof outline:
 - Assume that M has just **one control state** q and acceptance is by empty stack. Reverse the construction of the CFL \iff PDA Lemma; for each transition create a corresponding grammar rule:
 - The stack symbols will become auxiliaries in the grammar. The initial stack symbol is S , the start symbol.
 - For each transition rule of the form $q, Z \rightarrow q, \epsilon \mid (\epsilon \mid \epsilon \mid \epsilon)$ introduce a production $Z \rightarrow \epsilon$
 - We claim that for any $x \in \Sigma^*$ $(q, x, S) \vdash^* (q, \epsilon, \epsilon)$ iff $S \Rightarrow^* x$. This requires an inductive proof (given later).

Proof that One State Suffices

- Assume WLOG that $M = (Q, \Sigma, \Gamma, q_0, Z_0, \{q_f\}, \delta)$ is a pda accepting by empty stack, where q_f is a single final state, from which M empties its stack. Construct a new pda $M' = (\{q\}, \Sigma, \Gamma', q, Z'_0, A', \delta')$ that accepts the same language:
 - $\Gamma' = Q \times \Sigma \times Q$
 - Each element of Γ' is written $\langle q, Z, q' \rangle$ but is really just a single symbol.
 - The initial stack symbol of M' is $Z'_0 = \langle q_0, Z_0, q_f \rangle$.
 - For each transition of M : $(q', B_1 B_2 B_3 \dots B_n) \rightarrow (q, \epsilon, Z)$ include as transitions of M' : $(\langle q, \langle q_0, B_1, q_1 \rangle \langle q_1, B_2, q_2 \rangle \dots \langle q_{n-1}, B_n, q_n \rangle \rangle \rightarrow (q, \epsilon, \langle q, Z, q_f \rangle)$ for every choice of $q_1, q_2, \dots, q_n \in Q$ (choices not necessarily distinct). [If $n = 0$, then $(\langle q, \langle q_0, B_1, q_n \rangle \rangle \rightarrow (q, \epsilon, \langle q, Z, q_n \rangle)$.
 - That M' accepts the same language as M needs to be proved by induction.

2nd Proof of the CFL \iff PDA Lemma: bottom-up or shift-reduce technique

- Assume L is a context-free language. Then L has a context-free grammar G .
- Create a pda that accepts by final state.
- For each terminal symbol a , create a transition: $q_0, \epsilon, \epsilon \rightarrow q_0, a$. These have the effect of **shifting** the input string onto the stack (and reversing it in the process).
- For each production $A \rightarrow X_1 X_2 X_3 \dots X_n$ create transitions: $q_0, \epsilon, X_n \rightarrow q_1, \epsilon$
 $q_1, \epsilon, X_{n-1} \rightarrow q_2, \epsilon$
 $q_2, \epsilon, X_{n-2} \rightarrow q_3, \epsilon$
 \dots
 $q_{n-1}, \epsilon, X_1 \rightarrow q_0, A$ where the q_i other than q_0 are distinct for each production.
- This pda simulates a **rightmost** derivation of an accepted input string.

Example of Bottom-Up

Production	Transitions	State Sequence
$S \rightarrow (T$	$q_0, \epsilon, T \rightarrow q_1, \epsilon$ $q_1, \epsilon, (\rightarrow q_0, S$	$((()) \mid q_0 \mid Z_0$ $((()) \mid q_0 \mid (Z_0$ $((()) \mid q_0 \mid ((Z_0$ $((()) \mid q_0 \mid))((Z_0$ $((()) \mid q_0 \mid))S(Z_0$
$T \rightarrow S)$	$q_0, \epsilon,) \rightarrow q_2, \epsilon$ $q_2, \epsilon, S \rightarrow q_0, T$	$((()) \mid q_0 \mid (S(Z_0$ $((()) \mid q_0 \mid))S(Z_0$ $((()) \mid q_3 \mid (S(Z_0$ $((()) \mid q_0 \mid))S(Z_0$ $((()) \mid q_4 \mid S(Z_0$
$S \rightarrow ()$	$q_0, \epsilon,) \rightarrow q_3, \epsilon$ $q_3, \epsilon, (\rightarrow q_0, S$	$((()) \mid q_0 \mid (S(Z_0$ $((()) \mid q_0 \mid))S(Z_0$ $((()) \mid q_1 \mid T(Z_0$ $((()) \mid q_1 \mid (Z_0$ $((()) \mid q_0 \mid S(Z_0$ $((()) \mid q_5 \mid Z_0$
$S \rightarrow SS$	$q_0, \epsilon, S \rightarrow q_4, \epsilon$ $q_4, \epsilon, S \rightarrow q_0, S$	$((()) \mid q_0 \mid (S(Z_0$ $((()) \mid q_0 \mid))S(Z_0$ $((()) \mid q_2 \mid S(Z_0$ $((()) \mid q_0 \mid T(Z_0$ $((()) \mid q_1 \mid (Z_0$ $((()) \mid q_0 \mid S(Z_0$ $((()) \mid q_5 \mid Z_0$
shift transitions	$q_0, \epsilon, \epsilon \rightarrow q_0, \epsilon$ for each ϵ $q_0, \epsilon, \epsilon \rightarrow q_0, \epsilon$	$((()) \mid q_0 \mid S(Z_0$ $((()) \mid q_5 \mid Z_0$
accept transitions	$q_0, \epsilon, S \rightarrow q_5, \epsilon$ $q_5, \epsilon, Z_0 \rightarrow q_8, Z_0$	$((()) \mid q_5 \mid Z_0$ $((()) \mid q_8 \mid Z_0$

Derivation: $S \rightarrow (T \rightarrow (S) \rightarrow (SS) \rightarrow (S()) \rightarrow ((())$