Finite state + unbounded stack

Pushdown automata:

Q: (finite) set of states

Σ: input alphabet $\quad \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Γ: stack alphabet $\quad \Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

usually $\Sigma \subset \Gamma$ but Γ may have additional symbols.

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}_f(Q \times \Gamma_\varepsilon^*)$

$\mathcal{P}_f$: finite powerset

$q_0 \in Q$ start state

$F \subseteq Q$ accept states

To describe transitions

$a, b \longrightarrow c$

$a \in \Sigma_\varepsilon, b, c \in \Gamma_\varepsilon$

The PDA sees 'a' in the input
sees 'b' on top of the stack
then it pops the stack & pushes c
onto the stack

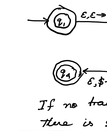
a may be ε : doesn't look at input

b may be ε : just push c on the stack
don't pop it

c may be ε : just pop the stack

PDA for $\{0^n 1^n \mid n \geq 0\}$ $\quad \Sigma = \{0,1\}$

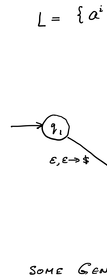
$\Gamma = \{0, 1, \$\}$



If no transition is indicated and there is still input; jam and reject.

$\Sigma = \{a, b, c\}$ $\quad \Gamma = \{a, b, c, \$\}$

$L = \{ a^i b^j c^k \mid i, j, k \geq 0;\ i = j \text{ OR } i = k \}$



SOME GENERAL REMARKS:

(i) Acceptance decisions only happen at the end of the input.

(ii) A PDA cannot decide to jam and reject when it has a valid move.

(iii) If there is a state with 2 or more moves and one of them is $\varepsilon, \varepsilon \to \varepsilon$ it can choose to do this at any time.

FUNDAMENTAL THEOREM:

Every CFL is recognized by a PDA.
Every language recognized by a PDA is a CFL

Idea 1 To show CFL's are recognized by PDA's : use the stack to keep track of partially generated strings
Use non-determinism to guess which rule to use.

Idea 2 For every pair of states define a new non-terminal & design a grammar to generate the strings that move the PDA between the states.

FACT The intersection of a CFL and a regular language is a CFL.

FACT Nondeterminism cannot be eliminated. The default def. of PDA includes non-determinism. There are special PDA's called deterministic PDA's or DPDA's which are less powerful than full PDA's.

FACT A CFL which can be recognized by a DPDA is called a deterministic CFL.

FACT Deterministic CFL's are closed under complementation even though CFL's in general are not.

Def of DPDA Transition fn

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow (Q \times \Gamma_\varepsilon^*) \cup \emptyset$

For every $q \in Q$, $a \in \Sigma_\varepsilon$, $x \in \Gamma$ exactly one of $\delta(q, a, x)$, $\delta(q, a, \varepsilon)$, $\delta(q, \varepsilon, x)$ and $\delta(q, \varepsilon, \varepsilon)$ is non-empty.

The first automaton I showed can be made into a DPDA by adding a dead state.

Recall unambiguous grammar:
Every DCFL has an unambiguous grammar. But not every language with an unambiguous grammar is a DCFL.

Programming languages are designed to be DCFL's so we have efficient algorithms to recognize whether a string is in the language.

We can have a different notion of acceptance for PDA's: No accept states
A string is accepted if the stack is empty at the end of the input.
This is equally powerful as acceptance by accept state. But obviously the machine cannot be the same.