On PDAs

1. Definitions of acceptance

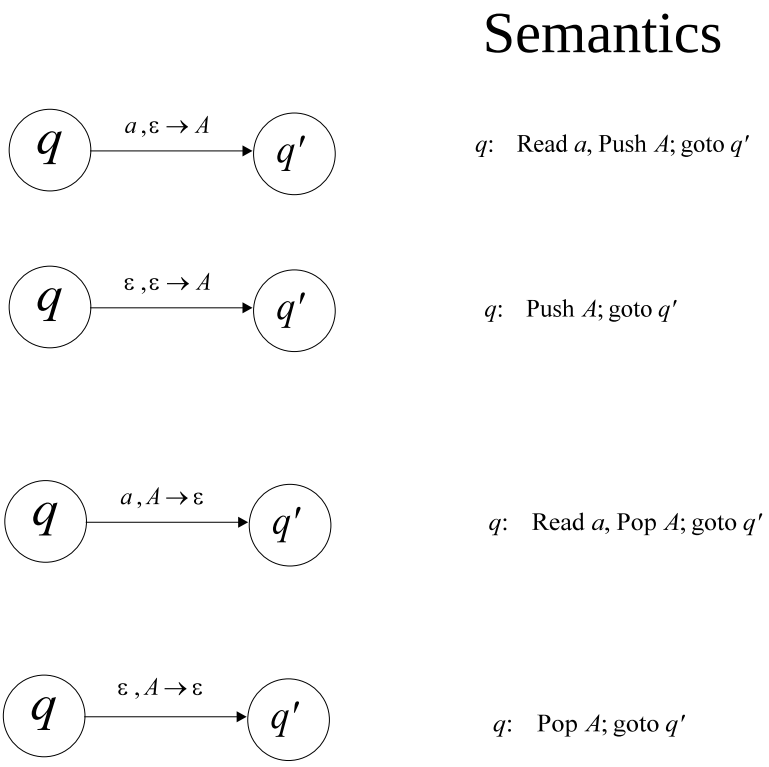
1.1 Definition. (PDA) A *pushdown automaton* or PDA, M, is essentially an NFA with a "stack". *Yes, the term "PDA" implies non determinism.*

That is,

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

Γ is new: A finite alphabet of stack symbols. It ***is*** allowed to have $\Sigma \cap \Gamma \neq \emptyset$. All the other ingredients of M have the same connotation as in NFAs.

The set of instructions in "coded" into the relation δ. We have only *four* types of instructions, given below in flow-diagram form:



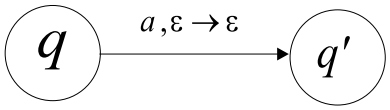
where, generically, an "A" denotes a member of Γ and an "a" denotes a member of the "tape (or "input") alphabet", Σ. The first and third are "mixed" instructions (both input and stack activity), while the second and fourth are "pure" (stack activity only). The 2nd is a "pure push instruction".

Mixed instructions "consume input" thus entailing a head move (to the right). Pure instructions do *not* entail (an input) head move. □

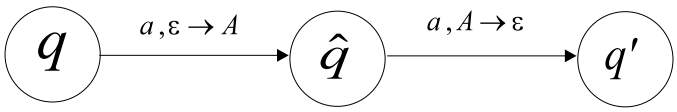
One realizes that the above "semantics" are just stated intentions, no more. Once we define "computations" the above semantics will become formal.

1.2 Example. In "the theory" (that is, when we study properties or limitations of PDAs, connections with CFGs, etc.) we will stick to the above four types of moves. In practice (i.e., when we masquerade as PDA programmers) we will allow the following two additional types of moves (really, 4 types, since we let $a \in \Sigma \cup \{\varepsilon\}$ below):

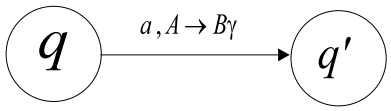
The first is an "ignore-the-stack" move (and also ignore the input, if $a = \varepsilon$). In the simulation, A is a *new* stack symbol and $\widehat{q}$ a *new* state. This means that if I "program" a PDA, M, with a "macro" like $a, \varepsilon \to \varepsilon$, then the expansion (implementation) of the macro is done by *adding* a new state $\widehat{q}$ and a new stack symbol A to the respective alphabets (Q and Γ) of my original M.



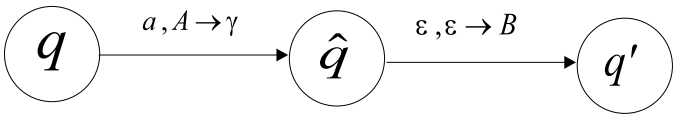
Simulated by



The next is also very useful when "programming" PDAs (i.e., constructing examples). The suggested implementation takes care of the general case in a compact (recursive) manner. It says that "if you know how to macro-expand (simulate) '$a, A \to \gamma$', then here is how to do '$a, A \to B\gamma$'". The "basis", when $\gamma = \varepsilon$, is one of our basic moves (Def. 1.1).



Simulated by



□

There are a number of concepts we need towards defining a *computation* of a pushdown automaton, and eventually string acceptance.

1.3 Definition. (Configurations) A *configuration* (or "snapshot" or *instantaneous description*—in short ID) of a PDA computation[†] is a triple (q, w, γ) where q denotes the current state, w is the *unspent* ("unexpended" or unprocessed, or, unread-so-far) input and γ is the total contents of the stack, read from top to bottom (that is, the leftmost symbol of γ is the topmost symbol of the stack).

The input head is at the first symbol of w, but ***that symbol has not been read yet!*** □

1.4 Definition. (Moves) If I and J are configurations of a PDA M, then $I \vdash J$—pronounced "I yields J"—means that ***there is a "move"***[‡] of M that transforms the ID I into J, *in one step*. The foregoing is "English" for the mathematically precise:

For all $a \in \Sigma \cup \{\varepsilon\}$, $y \in \Sigma^*$ and $\gamma \in \Gamma^*$,

$$(q, ay, A\gamma) \vdash (q', y, \gamma)$$

iff instruction 3 (Def. 1.1, case $a \in \Sigma$), or 4 (Def. 1.1, case $a = \varepsilon$) is available.

For all $a \in \Sigma \cup \{\varepsilon\}$, $y \in \Sigma^*$ and $\gamma \in \Gamma^*$,

$$(q, ay, \gamma) \vdash (q', y, A\gamma)$$

iff instruction 1 (Def. 1.1, case $a \in \Sigma$), or 2 (Def. 1.1, case $a = \varepsilon$) is available. □

[†]You will not miss the fact that we do *not* need to know what a "computation" is before we know what an "ID" is. Indeed we will define a computation as an appropriate sequence of IDs.

[‡]Note: ***there is***. I am *not* saying that it is the *only* move, or that I uniquely determines J.

1.5 Definition. (Initial, Terminal IDs) A configuration is ***initial*** iff it has the form (q_0, x, ε). This captures the intended semantics that computations with *input* x start at state "q_0" (generic initial state) and with an empty stack.

A configuration (q, ε, γ) is *terminal* or *final* (I did *not* say "accepting!) iff there is *no* defined next move. That is,

$$\neg(\exists J)\Big((q, \varepsilon, \gamma) \vdash J\Big)$$

□

1.6 Definition. (PDA computations) A PDA computation is a sequence of IDs, $I_0, \ldots, I_n$ such that

(1) I_0 is initial

(2) I_n is terminal (final)

(3) For $i = 0, \ldots, n-1$, $I_i \vdash I_{i+1}$.

As usual, we write $I_0 \vdash^* I_n$ (denoting 0 or more occurrences of "$\vdash$") and say " $I_0 \vdash^* I_n$ is a computation", which is a slight abuse of language: We should have said—if we do not want to mention $I_1, \ldots, I_{n-1}$—that "there is a computation with I_0 as initial and I_n as terminal IDs". □

1.7 Remark. Thus, we require our computations to be terminating. This is implicit in PDA models that one sees in the literature. For example, in the approach taken in Hopcroft and Ullman, a stack move $\varepsilon \to A$ (push) is *not* allowed. Instead, the only stack moves allowed are $A \to \gamma$ (in English, "the stack must always be consulted *before* moving ahead").

Thus, in particular, Hopcroft and Ullman PDAs that accept by "empty stack" end up in a terminating configuration, since no move is permitted without consulting the stack.[†]

1.8 Definition. (ES, FS, and ES+FS acceptance)
The string x ***is ES-accepted*** [‡] by a PDA, M, iff there is a computation $(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, \varepsilon)$ for some $q \in Q$.

For ES acceptance, the set F of final states is irrelevant, and is often taken to be $\emptyset$.

The string x ***is FS-accepted*** [§] by a PDA, M, iff there is a computation $(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, \gamma)$ for some $q \in F$. The stack contents, γ, at computation's end is irrelevant to FS-acceptance.

[†]To allow such PDAs to *begin* a computation, the stack is "externally"—i.e., *not* by a PDA instruction—initialized with a special "bottom of the stack 'initial' symbol". Kozen—a book I used a few sessions back—has similar conventions.

[‡]By Empty Stack.

[§]By Final State.

The string x ***is ES+FS-accepted*** [†] by a PDA, M, iff there is a computation $(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, \varepsilon)$ for some $q \in F$. ***This is the way Sipser, and Lewis and Papadimitriou***—another book that I used way back—***want their PDA accept a string.***

If the context helps, we may simply say M accepts x, without having to say in what mode (ES, FS, or ES+FS). □

1.9 Definition. (Recognition of Languages) If M is a PDA, then $L(M)$ denotes the set $\{x \in \Sigma^* : x \text{ is accepted by } M\}$ (by ES, FS, or ES+FS acceptance, as the case may be).

"L is ES- (FS-, or ES+FS-) *recognized*", if there is a PDA M that accepts by ES (respectively, FS or ES+FS) such that $L = L(M)$. □

2. ES vs FS vs ES+FS

2.1 Theorem. (FS can simulate ES) *If M is a PDA accepting by ES, then we can construct a PDA N that accepts by FS, so that $L(M) = L(N)$.*

Proof. We refer to the figure below. Let

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

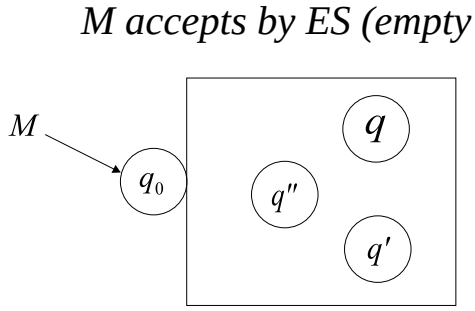
where $F = \emptyset$, be our ES-PDA. We build N as it is (partially) suggested in the figure below.

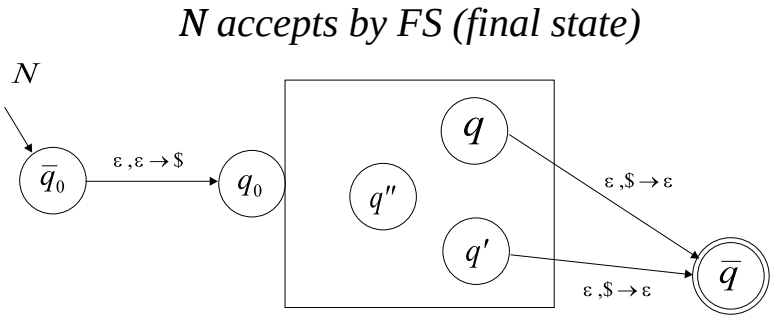
$$N = (\Sigma, \overline{\Gamma}, \overline{Q}, \overline{\delta}, \overline{q_0}, \overline{F})$$

where $\overline{Q} = Q \cup \{\overline{q_0}, \overline{q}\}$, $\overline{F} = \{\overline{q}\}$, and $\overline{\Gamma} = \Gamma \cup \{\$\}$. $\overline{\delta}$ is δ, augmented by the *new* instructions pictured below. The one involving $\overline{q_0}$ is self explanatory. The "pure pop" instructions that lead to $\overline{q}$

$$\textit{are only defined on states } q \textit{ that have } \textbf{\textit{no pure push moves}} \qquad (1)$$

[†] By Empty Stack *and* Final State. We may also say "FS+ES accepted"





We now proceed to prove that

$$L(M) = L(N) \tag{2}$$

For $\subseteq$: Let $x \in L(M)$. Then

$$(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, \varepsilon) \text{ is an } M\text{-computation}^\dagger$$

Thus,

$$(\overline{q_0}, x, \varepsilon) \vdash (q_0, x, \$) \vdash^* (q, \varepsilon, \$) \vdash (\overline{q}, \varepsilon, \varepsilon) \text{ is an } N\text{-computation}$$

Clearly the last ID is terminating, since, by construction of N, $\overline{q}$ has no moves at all.

Let us justify the last "$\vdash$": Since $(q, \varepsilon, \varepsilon)$ is terminal in M, q cannot have pure push moves (else the computation *could* continue from $(q, \varepsilon, \varepsilon)$—the latter ID would not then be terminal!) Thus, this q ***is*** connected to $\overline{q}$ as shown in the figure above.

In short, $x \in L(N)$.

For $\supseteq$: Let $x \in L(N)$. Then

$$(\overline{q_0}, x, \varepsilon) \vdash^* (\overline{q}, \varepsilon, \gamma) \text{ is an } N\text{-computation} \tag{3}$$

The ***only way*** to reach $\overline{q}$ is to reach it—in one move—from some q (of the original M) that has no pure push moves (in M) (see figure above). Thus, (3),

[†]This *includes* the assertion that $(q, \varepsilon, \varepsilon)$ is terminal—see Def. 1.5.

in some more detail, is the N-computation below, where the first step is obvious (and inevitable) due to the construction of N:

$$(\overline{q_0}, x, \varepsilon) \vdash (q_0, x, \$) \vdash^* (q, \varepsilon, \gamma') \vdash (\overline{q}, \varepsilon, \gamma) \tag{4}$$

Now, the only way for the last $\vdash$ to be valid (see Def. 1.4 and the construction of N) is that $\gamma' = \$\gamma$. Moreover, since N can only write \$ once, ***in the very first move*** (and the M-part cannot write \$ at all), we conclude that $\gamma = \varepsilon$.

Now we see (4) more clearly:

$$(\overline{q_0}, x, \varepsilon) \vdash (q_0, x, \$) \vdash^* (q, \varepsilon, \$) \vdash (\overline{q}, \varepsilon, \varepsilon)$$

Thus, forgetting first and last moves, $(q_0, x, \$) \vdash^* (q, \varepsilon, \$)$ and hence[†]

$$(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, \varepsilon) \text{ is an } M\text{-computation} \tag{5}$$

Note that ID $(q, \varepsilon, \varepsilon)$ is terminal in M. Thus, since M is an ES machine, $x \in L(M)$. □

2.2 Theorem. (ES can simulate FS) *If M is a PDA accepting by FS, then we can construct a PDA N that accepts by ES, so that $L(M) = L(N)$.*

Proof. We refer to the figure below. Let

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

be our FS-PDA. Without loss of generality we assume that $F = \{q\}$ and that q has no moves. Indeed, if it is not so designed, then do:

(i) Add new a state q and make it final.

(ii) For each (original) final state q' add a move "$\varepsilon, \varepsilon \to \varepsilon$" (recall our macros!) from q' to q.

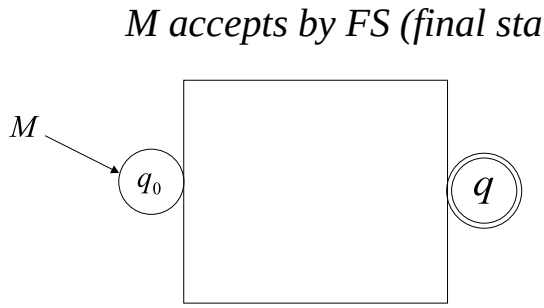
(iii) Give ***no moves*** to q.

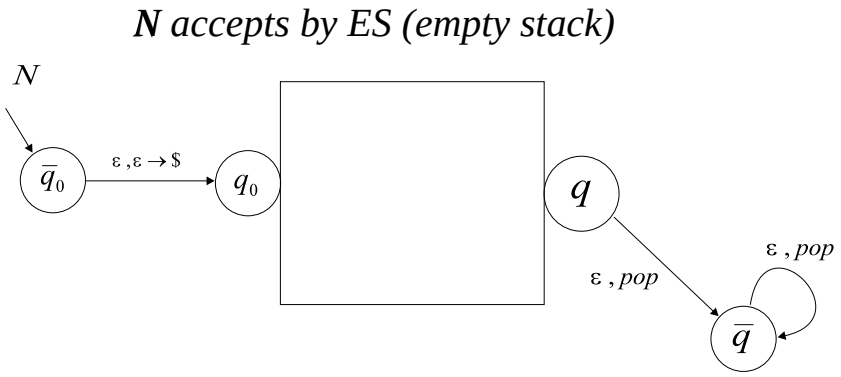
We build N as it is (partially) suggested in the figure below.

$$N = (\Sigma, \overline{\Gamma}, \overline{Q}, \overline{\delta}, \overline{q_0}, \overline{F})$$

where $\overline{Q} = Q \cup \{\overline{q_0}, \overline{q}\}$, $\overline{F} = \emptyset$ (thus, we have removed accept-status from q), and $\overline{\Gamma} = \Gamma \cup \{\$\}$. $\overline{\delta}$ is δ, augmented by the *new* instructions pictured below.

[†]In $(q_0, x, \$) \vdash^* (q, \varepsilon, \$)$, that is, in $(q_0, x, \$) \vdash I_1 \vdash I_2 \vdash \cdots \vdash I_r \vdash (q, \varepsilon, \$)$ each I_i has \$ at the bottom of its stack. In (5) we left all else the same, but we removed all the \$'s.





We now prove that $L(M) = L(N)$.

For $\subseteq$: Let $x \in L(M)$. Then

$$(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, \gamma) \text{ is an } M\text{-computation}^\dagger$$

Then, trivially,

$$(\overline{q_0}, x, \varepsilon) \vdash (q_0, x, \$) \vdash^* (q, \varepsilon, \gamma\$) \vdash^* (\overline{q}, \varepsilon, \varepsilon) \text{ is an } N\text{-computation}^\ddagger$$

Thus, $x \in L(N)$.

For $\supseteq$: Let $x \in L(N)$. Then

$$(\overline{q_0}, x, \varepsilon) \vdash^* (\overline{q}, \varepsilon, \varepsilon) \text{ is an } N\text{-computation} \tag{1}$$

Pause. Wait a minute! Why is $\overline{q}$ the state in the last ID of (1)? Well, because ***no other state*** can ponder an empty stack. The only states that can erase \$ are q and $\overline{q}$, ***each by a move that leads to*** $\overline{q}$.

As in the proof of the previous theorem, let us work back from the end: $\overline{q}$ consumes no input and is only accessible from state q (see figure). Thus, when q sent the computation to $\overline{q}$ (by popping once), ***the input must have***

$^\dagger(q, \varepsilon, \varepsilon)$ is terminal, as required, due to the "without loss of generality" discussion above.
‡Terminating, since $\overline{q}$ has no pure push moves.

been already consumed. This, coupled with the inevitable ***first*** move that places a \$ in the stack—a symbol ***which cannot be erased by "internal"*** M***-moves***—gives a more "detailed picture" of (1), below:

$$(\overline{q_0}, x, \varepsilon) \vdash (q_0, x, \$) \vdash^* (q, \varepsilon, \gamma\$) \vdash^* (\overline{q}, \varepsilon, \varepsilon)$$

Now the part $(q_0, x, \$) \vdash^* (q, \varepsilon, \gamma\$)$, stripped, throughout, of the bottom \$ stack symbol becomes the (***terminating***—remember the fact that q has no moves in M!) computation

$$(q_0, x, \varepsilon) \vdash^* (q, \varepsilon, \gamma)$$

Since $q \in F$, it follows that $x \in L(M)$. □

2.3 Corollary. *All three acceptance modes are equivalent.*

Proof. By Theorems 2.1 and 2.2, ES and FS acceptance modes are equivalent. What about ES+FS?

Well, if M is ES, then an N that accepts by ES+FS can be build that simulates M. We achieved this (without fanfare, until now) in the proof of Theorem 2.1.

Conversely, let M accept by ES+FS. I can then build an ES-PDA simulator, N. The construction and proof is exactly as in Theorem 2.2 since neither the construction nor the proof there made ***any assumptions whatsoever*** on what the γ (that was left in the stack at the end of the computation) was. □