



Pushdown automaton

In the theory of computation, a branch of theoretical computer science, a **pushdown automaton (PDA)** is a type of automaton that employs a stack.

Pushdown automata are used in theories about what can be computed by machines. They are more capable than finite-state machines but less capable than Turing machines (see below). Deterministic pushdown automata can recognize all deterministic context-free languages while nondeterministic ones can recognize all context-free languages, with the former often used in parser design.

The term "pushdown" refers to the fact that the stack can be regarded as being "pushed down" like a tray dispenser at a cafeteria, since the operations never work on elements other than the top element. A **stack automaton**, by contrast, does allow access to and operations on deeper elements. Stack automata can recognize a strictly larger set of languages than pushdown automata.^[1] A nested stack automaton allows full access, and also allows stacked values to be entire sub-stacks rather than just single finite symbols.

Informal description

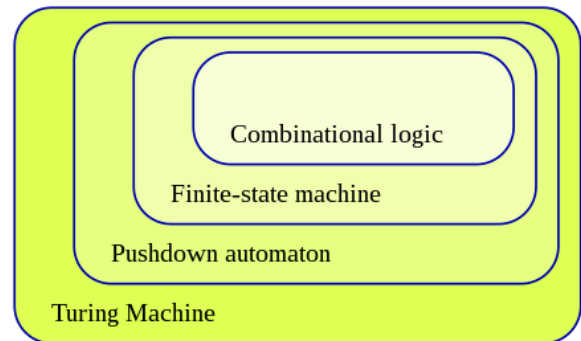
A finite-state machine just looks at the input signal and the current state: it has no stack to work with. It chooses a new state, the result of following the transition. A **pushdown automaton (PDA)** differs from a finite state machine in two ways:

1. It can use the top of the stack to decide which transition to take.
2. It can manipulate the stack as part of performing a transition.

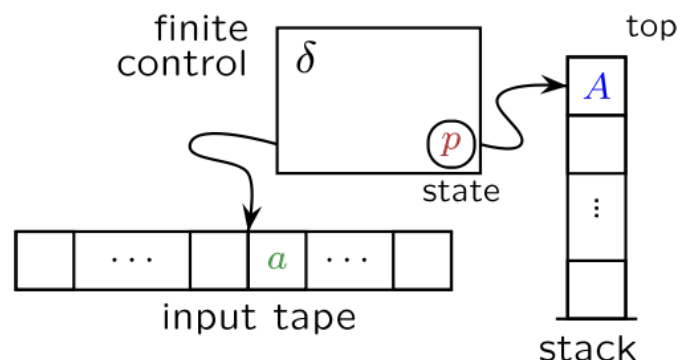
A pushdown automaton reads a given input string from left to right. In each step, it chooses a transition by indexing a table by input symbol, current state, and the symbol at the top of the stack. A pushdown automaton can also manipulate the stack, as part of performing a transition. The manipulation can be to push a particular symbol to the top of the stack, or to pop off the top of the stack. The automaton can alternatively ignore the stack, and leave it as it is.

Put together: Given an input symbol, current state, and stack symbol, the automaton can follow a transition to another state, and optionally manipulate (push or pop) the stack.

Automata theory



Classes of automata



A diagram of a pushdown automaton

If, in every situation, at most one such transition action is possible, then the automaton is called a **deterministic pushdown automaton (DPDA)**. In general, if several actions are possible, then the automaton is called a **general**, or **nondeterministic**, **PDA**. A given input string may drive a nondeterministic pushdown automaton to one of several configuration sequences; if one of them leads to an accepting configuration after reading the complete input string, the latter is said to belong to the *language accepted by the automaton*.

Formal definition

We use standard formal language notation: Γ^* denotes the set of finite-length strings over alphabet Γ and ϵ denotes the empty string.

A PDA is formally defined as a 7-tuple:

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

- Q is a finite set of *states*
- Σ is a finite set which is called the *input alphabet*
- Γ is a finite set which is called the *stack alphabet*
- δ is a finite subset of $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$, the *transition relation*
- $q_0 \in Q$ is the *start state*
- $Z \in \Gamma$ is the *initial stack symbol*
- $F \subseteq Q$ is the set of *accepting states*

An element $(p, a, A, q, \alpha) \in \delta$ is a transition of M . It has the intended meaning that M , in state $p \in Q$, on the input $a \in \Sigma \cup \{\epsilon\}$ and with $A \in \Gamma$ as topmost stack symbol, may read a , change the state to q , pop A , replacing it by pushing $\alpha \in \Gamma^*$. The $(\Sigma \cup \{\epsilon\})$ component of the transition relation is used to formalize that the PDA can either read a letter from the input, or proceed leaving the input untouched.

In many texts^{[2]:110} the transition relation is replaced by an (equivalent) formalization, where

- δ is the *transition function*, mapping $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ into finite subsets of $Q \times \Gamma^*$

Here $\delta(p, a, A)$ contains all possible actions in state p with A on the stack, while reading a on the input. One writes for example $\delta(p, a, A) = \{(q, BA)\}$ precisely when $(q, BA) \in \{(q, BA)\}$, $(q, BA) \in \delta(p, a, A)$, because $((p, a, A), \{(q, BA)\}) \in \delta$. Note that *finite* in this definition is essential.

Computations

In order to formalize the semantics of the pushdown automaton a description of the current situation is introduced. Any 3-tuple $(p, w, \beta) \in Q \times \Sigma^* \times \Gamma^*$ is called an instantaneous description (ID) of M , which includes the current state, the part of the input tape that has not been read, and the contents of the stack (topmost symbol written first). The transition relation δ defines the step-relation \vdash_M of M on instantaneous descriptions. For instruction $(p, a, A, q, \alpha) \in \delta$ there exists a step $(p, ax, A\gamma) \vdash_M (q, x, \alpha\gamma)$, for every $x \in \Sigma^*$ and every $\gamma \in \Gamma^*$.

In general pushdown automata are nondeterministic meaning that in a given instantaneous description (p, w, β) there may be several possible steps. Any of these steps can be chosen in a computation. With the

above definition in each step always a single symbol (top of the stack) is popped, replacing it with as many symbols as necessary. As a consequence no step is defined when the stack is empty.

Computations of the pushdown automaton are sequences of steps. The computation starts in the initial state q_0 with the initial stack symbol Z on the stack, and a string w on the input tape, thus with initial description (q_0, w, Z) . There are two modes of accepting. The pushdown automaton either accepts by final state, which means after reading its input the automaton reaches an accepting state (in F), or it accepts by empty stack (ϵ), which means after reading its input the automaton empties its stack. The first acceptance mode uses the internal memory (state), the second the external memory (stack).

Formally one defines

1. $L(M) = \{w \in \Sigma^* \mid (q_0, w, Z) \vdash_M^* (f, \epsilon, \gamma) \text{ with } f \in F \text{ and } \gamma \in \Gamma^*\}$
(final state)
2. $N(M) = \{w \in \Sigma^* \mid (q_0, w, Z) \vdash_M^* (q, \epsilon, \epsilon) \text{ with } q \in Q\}$ (empty stack)

Here \vdash_M^* represents the reflexive and transitive closure of the step relation \vdash_M meaning any number of consecutive steps (zero, one or more).

For each single pushdown automaton these two languages need to have no relation: they may be equal but usually this is not the case. A specification of the automaton should also include the intended mode of acceptance. Taken over all pushdown automata both acceptance conditions define the same family of languages.

Theorem. For each pushdown automaton M one may construct a pushdown automaton M' such that $L(M) = N(M')$, and vice versa, for each pushdown automaton M one may construct a pushdown automaton M' such that $N(M) = L(M')$

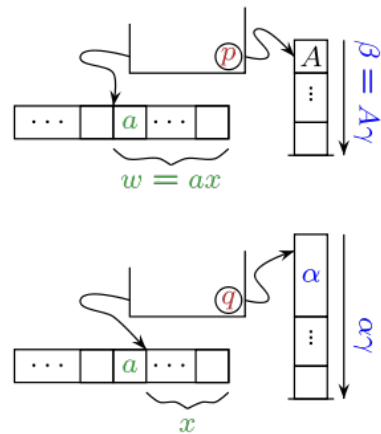
Example

The following is the formal description of the PDA which recognizes the language $\{0^n 1^n \mid n \geq 0\}$ by final state:

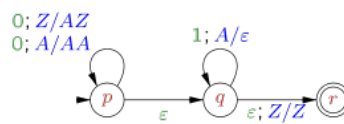
$M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where

- **states:** $Q = \{p, q, r\}$
- **input alphabet:** $\Sigma = \{0, 1\}$
- **stack alphabet:** $\Gamma = \{A, Z\}$
- **start state:** $q_0 = p$
- **start stack symbol:** Z
- **accepting states:** $F = \{r\}$

The transition relation δ consists of the following six instructions:



a step of the pushdown automaton



PDA for $\{0^n 1^n \mid n \geq 0\}$
(by final state)

$(p, 0, Z, p, AZ),$
 $(p, 0, A, p, AA),$
 $(p, \epsilon, Z, q, Z),$
 $(p, \epsilon, A, q, A),$
 $(q, 1, A, q, \epsilon),$ and
 $(q, \epsilon, Z, r, Z).$

In words, the first two instructions say that in state p any time the symbol 0 is read, one A is pushed onto the stack. Pushing symbol A on top of another A is formalized as replacing top A by AA (and similarly for pushing symbol A on top of a Z).

The third and fourth instructions say that, at any moment the automaton may move from state p to state q .

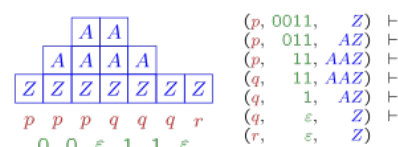
The fifth instruction says that in state q , for each symbol 1 read, one A is popped.

Finally, the sixth instruction says that the machine may move from state q to accepting state r only when the stack consists of a single Z .

There seems to be no generally used representation for PDA. Here we have depicted the instruction (p, a, A, q, α) by an edge from state p to state q labelled by $a; A/\alpha$ (read a ; replace A by α).

Understanding the computation process

The following illustrates how the above PDA computes on different input strings. The subscript M from the step symbol \vdash is here omitted.



accepting computation for
0011

- a. Input string = 0011. There are various computations, depending on the moment the move from state p to state q is made. Only one of these is accepting.
 - i. $(p, 0011, Z) \vdash (q, 0011, Z) \vdash (r, 0011, Z)$
The final state is accepting, but the input is not accepted this way as it has not been read.
 - ii. $(p, 0011, Z) \vdash (p, 011, AZ) \vdash (q, 011, AZ)$
No further steps possible.
 - iii. $(p, 0011, Z) \vdash (p, 011, AZ) \vdash (p, 11, AAZ) \vdash (q, 11, AAZ) \vdash (q, 1, AZ) \vdash (q, \epsilon, Z) \vdash (r, \epsilon, Z)$
Accepting computation: ends in accepting state, while complete input has been read.
- b. Input string = 00111. Again there are various computations. None of these is accepting.
 - i. $(p, 00111, Z) \vdash (q, 00111, Z) \vdash (r, 00111, Z)$
The final state is accepting, but the input is not accepted this way as it has not been read.
 - ii. $(p, 00111, Z) \vdash (p, 0111, AZ) \vdash (q, 0111, AZ)$
No further steps possible.
 - iii. $(p, 00111, Z) \vdash (p, 0111, AZ) \vdash (p, 111, AAZ) \vdash (q, 111, AAZ) \vdash (q, 11, AZ) \vdash (q, 1, Z) \vdash (r, 1, Z)$
The final state is accepting, but the input is not accepted this way as it has not

been (completely) read.

PDA and context-free languages

Every context-free grammar can be transformed into an equivalent nondeterministic pushdown automaton. The derivation process of the grammar is simulated in a leftmost way. Where the grammar rewrites a nonterminal, the PDA takes the topmost nonterminal from its stack and replaces it by the right-hand part of a grammatical rule (*expand*). Where the grammar generates a terminal symbol, the PDA reads a symbol from input when it is the topmost symbol on the stack (*match*). In a sense the stack of the PDA contains the unprocessed data of the grammar, corresponding to a pre-order traversal of a derivation tree.

Technically, given a context-free grammar, the PDA has a single state, 1, and its transition relation is constructed as follows.

1. $(1, \epsilon, A, 1, \alpha)$ for each rule $A \rightarrow \alpha$ (*expand*)
2. $(1, a, a, 1, \epsilon)$ for each terminal symbol a (*match*)

The PDA accepts by empty stack. Its initial stack symbol is the grammar's start symbol.

For a context-free grammar in Greibach normal form, defining $(1, \gamma) \in \delta(1, a, A)$ for each grammar rule $A \rightarrow a\gamma$ also yields an equivalent nondeterministic pushdown automaton.^{[2]:115}

The converse, finding a grammar for a given PDA, is not that easy. The trick is to code two states of the PDA into the nonterminals of the grammar.

Theorem. For each pushdown automaton M one may construct a context-free grammar G such that $N(M) = L(G)$.^{[2]:116}

The language of strings accepted by a deterministic pushdown automaton (DPDA) is called a deterministic context-free language. Not all context-free languages are deterministic.^[note 1] As a consequence, the DPDA is a strictly weaker variant of the PDA. Even for regular languages, there is a size explosion problem: for any recursive function f and for arbitrarily large integers n , there is a PDA of size n describing a regular language whose smallest DPDA has at least $f(n)$ states.^[4] For many non-regular PDAs, any equivalent DPDA would require an unbounded number of states.

A finite automaton with access to two stacks is a more powerful device, equivalent in power to a Turing machine.^{[2]:171} A linear bounded automaton is a device which is more powerful than a pushdown automaton but less so than a Turing machine.^[note 2]

PDA and Turing machines

A pushdown automaton is computationally equivalent to a 'restricted' Turing Machine (TM) with two tapes which is restricted in the following manner- On the first tape, the TM can only read the input and move from left to right (it cannot make changes). On the second tape, it can only 'push' and 'pop' data. Or equivalently, it can read, write and move left and right with the restriction that the only action it can perform at each step is to either delete the left-most character in the string (pop) or add an extra character left to the left-most character in the string (push).

That a PDA is weaker than a TM can be brought down to the fact that the procedure 'pop' deletes some

data. In order to make a PDA as strong as a TM, we need to save somewhere the data lost through 'pop'. We can achieve this by introducing a second stack. In the TM model of PDA of last paragraph, this is equivalent to a TM with 3 tapes, where the first tape is the read-only input tape, and the 2nd and the 3rd tape are the 'push and pop' (stack) tapes. In order for such a PDA to simulate any given TM, we give the input of the PDA to the first tape, while keeping both the stacks empty. It then goes on to push all the input from the input tape to the first stack. When the entire input is transferred to the 1st stack, now we proceed like a normal TM, where moving right on the tape is the same as popping a symbol from the 1st stack and pushing a (possibly updated) symbol into the second stack, and moving left corresponds to popping a symbol from the 2nd stack and pushing (a possibly updated) symbol into the first stack. We hence have a PDA with 2 stacks that can simulate any TM.

Generalized pushdown automaton (GPDA)

A GPDA is a PDA that writes an entire string of some known length to the stack or removes an entire string from the stack in one step.

A GPDA is formally defined as a 6-tuple:

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

where Q, Σ, Γ, q_0 , and F are defined the same way as a PDA.

$$\delta: Q \times \Sigma_\epsilon \times \Gamma^* \longrightarrow P(Q \times \Gamma^*)$$

is the transition function.

Computation rules for a GPDA are the same as a PDA except that the a_{i+1} 's and b_{i+1} 's are now strings instead of symbols.

GPDA's and PDA's are equivalent in that if a language is recognized by a PDA, it is also recognized by a GPDA and vice versa.

One can formulate an analytic proof for the equivalence of GPDA's and PDA's using the following simulation:

Let $\delta(q_1, w, x_1 x_2 \dots x_m) \longrightarrow (q_2, y_1 y_2 \dots y_n)$ be a transition of the GPDA

where $q_1, q_2 \in Q, w \in \Sigma_\epsilon, x_1, x_2, \dots, x_m \in \Gamma^*, m \geq 0, y_1, y_2, \dots, y_n \in \Gamma^*, n \geq 0$.

Construct the following transitions for the PDA:

$$\begin{array}{ll} \delta'(q_1, w, x_1) & \longrightarrow (p_1, \epsilon) \\ \delta'(p_1, \epsilon, x_2) & \longrightarrow (p_2, \epsilon) \\ & \vdots \\ \delta'(p_{m-1}, \epsilon, x_m) & \longrightarrow (p_m, \epsilon) \\ \delta'(p_m, \epsilon, \epsilon) & \longrightarrow (p_{m+1}, y_n) \\ \delta'(p_{m+1}, \epsilon, \epsilon) & \longrightarrow (p_{m+2}, y_{n-1}) \\ & \vdots \\ \delta'(p_{m+n-1}, \epsilon, \epsilon) & \longrightarrow (q_2, y_1). \end{array}$$

Stack automaton

As a generalization of pushdown automata, Ginsburg, Greibach, and Harrison (1967) investigated **stack automata**, which may additionally step left or right in the input string (surrounded by special endmarker symbols to prevent slipping out), and step up or down in the stack in read-only mode.^{[5][6]} A stack automaton is called *nonerasing* if it never pops from the stack. The class of languages accepted by nondeterministic, nonerasing stack automata is *NSPACE*(n^2), which is a superset of the context-sensitive languages.^[1] The class of languages accepted by deterministic, nonerasing stack automata is *DSPACE*($n \cdot \log(n)$).^[1]

Alternating pushdown automata

An **alternating pushdown automaton** (APDA) is a pushdown automaton with a state set

- $Q = Q_{\exists} \cup Q_{\forall}$ where $Q_{\exists} \cap Q_{\forall} = \emptyset$.

States in Q_{\exists} and Q_{\forall} are called *existential* resp. *universal*. In an existential state an APDA nondeterministically chooses the next state and accepts if *at least one* of the resulting computations accepts. In a universal state APDA moves to all next states and accepts if *all* the resulting computations accept.

The model was introduced by Chandra, Kozen and Stockmeyer.^[7] Ladner, Lipton and Stockmeyer^[8] proved that this model is equivalent to EXPTIME i.e. a language is accepted by some APDA if, and only if, it can be decided by an exponential-time algorithm.

Aizikowitz and Kaminski^[9] introduced *synchronized alternating pushdown automata* (SAPDA) that are equivalent to conjunctive grammars in the same way as nondeterministic PDA are equivalent to context-free grammars.

See also

- Stack machine
- Context-free grammar
- Finite automaton
- Counter automaton
- Queue automaton

Notes

1. The set of even-length palindromes of bits can't be recognized by a deterministic PDA, but is a context-free language, with the grammar $S \rightarrow \varepsilon \mid 0S0 \mid 1S1$.^[3]
2. Linear bounded automata are acceptors for the class of context-sensitive languages,^{[2]:225} which is a proper superclass of the context-free languages, and a proper subclass of Turing-recognizable (i.e. recursively enumerable) languages.^{[2]:228}

References

1. John E. Hopcroft; Jeffrey D. Ullman (1967). "Nonerasing Stack Automata" (<https://doi.org/10.1016%2Fs0022-0000%2867%2980013-8>). *Journal of Computer and System Sciences*. **1** (2): 166–186. doi:10.1016/s0022-0000(67)80013-8 (<https://doi.org/10.1016%2Fs0022-0000%2867%2980013-8>).
2. John E. Hopcroft and Jeffrey D. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation* (<https://archive.org/details/introductiontoau00hopc>). Reading/MA: Addison-Wesley. ISBN 0-201-02988-X.
3. John E. Hopcroft; Rajeev Motwani; Jeffrey D. Ullman (2003). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley. Here: Sect.6.4.3, p.249
4. Holzer, Markus; Kutrib, Martin (2019). "Non-Recursive Trade-Offs Are "Almost Everywhere" ". *Computing with Foresight and Industry*. **11558**: 25–36. doi:10.1007/978-3-030-22996-2_3 (https://doi.org/10.1007%2F978-3-030-22996-2_3). This follows from the quoted [22, Proposition 7] and the stated observation that any deterministic pushdown automaton can be converted into an equivalent finite automaton of at most doubly-exponential size.
5. Seymour Ginsburg, Sheila A. Greibach and Michael A. Harrison (1967). "Stack Automata and Compiling" (<https://doi.org/10.1145%2F321371.321385>). *J. ACM*. **14** (1): 172–201. doi:10.1145/321371.321385 (<https://doi.org/10.1145%2F321371.321385>).
6. Seymour Ginsburg, Sheila A. Greibach and Michael A. Harrison (1967). "One-Way Stack Automata". *J. ACM*. **14** (2): 389–418. doi:10.1145/321386.321403 (<https://doi.org/10.1145%2F321386.321403>).
7. Chandra, Ashok K.; Kozen, Dexter C.; Stockmeyer, Larry J. (1981). "Alternation". *Journal of the ACM*. **28** (1): 114–133. doi:10.1145/322234.322243 (<https://doi.org/10.1145%2F322234.322243>). ISSN 0004-5411 (<https://www.worldcat.org/issn/0004-5411>).
8. Ladner, Richard E.; Lipton, Richard J.; Stockmeyer, Larry J. (1984). "Alternating Pushdown and Stack Automata". *SIAM Journal on Computing*. **13** (1): 135–155. doi:10.1137/0213010 (<https://doi.org/10.1137%2F0213010>). ISSN 0097-5397 (<https://www.worldcat.org/issn/0097-5397>).
9. Aizikowitz, Tamar; Kaminski, Michael (2011). "LR(0) Conjunctive Grammars and Deterministic Synchronized Alternating Pushdown Automata". *Computer Science - Theory and Applications*. Lecture Notes in Computer Science. Vol. 6651. pp. 345–358. doi:10.1007/978-3-642-20712-9_27 (https://doi.org/10.1007%2F978-3-642-20712-9_27). ISBN 978-3-642-20711-2. ISSN 0302-9743 (<https://www.worldcat.org/issn/0302-9743>).
 - Michael Sipser (1997). *Introduction to the Theory of Computation* (<https://archive.org/details/introductiontoth00sips>). PWS Publishing. ISBN 0-534-94728-X. Section 2.2: Pushdown Automata, pp. 101–114.
 - Jean-Michel Autebert, Jean Berstel, Luc Boasson, *Context-Free Languages and Push-Down Automata* (<http://www-igm.univ-mlv.fr/~berstel/Articles/1997CFLPDA.pdf>), in: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, Vol. 1, Springer-Verlag, 1997, 111–174.

External links

- JFLAP (<https://www.jflap.org>), simulator for several types of automata including

nondeterministic pushdown automata

- [CoAn \(https://www.elstel.org/coan\)](https://www.elstel.org/coan), another simulator for several machine types including nondeterministic pushdown automata (C++, Windows, Linux, MacOS)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Pushdown_automaton&oldid=1136388051"