Two-Stack Deterministic Pushdown Automata

Wesly Samson
2401 Taft Avenue
Manila, Philippines
wesly_samson_@dlsu.edu.ph

Rafael Antonio Austria
2401 Taft Avenue
Manila, Philippines
rafael_austria@dlsu.edu.ph

Reign Elaiza Larraquel
2401 Taft Avenue
Manila, Philippines
reign_larraquel@dlsu.edu.ph

ABSTRACT

This case study delves into the exploration of the Two Stack Deterministic Pushdown Automata, a model of computation that extends the traditional one-stack pushdown automata, with an additional independent stack. It is defined as a 7-tuple with an added second stack manipulation on the transitions. Moreover, the transitions of a two-stack deterministic PDA must be all unique. It also has two ways to accept a string: empty-stack basis or final-state basis. In terms of its formal language and computational power, two-stack deterministic PDAs are stronger than deterministic finite automata. However, they are weaker than both two-stack non-deterministic PDA and Turing machines. In terms of the Chomsky hierarchy, two-stack deterministic PDAs fall somewhere between recursively enumerable (type-0) and context-sensitive (type-1) languages. It was also found out that a two-stack deterministic PDA can't always recognize regular, context-free, or context-sensitive languages due to its deterministic limitations. In terms of its real-life application, a two-stack deterministic PDA can also recognize deterministic patterns while utilizing its two stacks to validate input strings. One example of this would be email validation with respect to the usual email validation regex.

Machine Definition

A one-way, two-stack deterministic pushdown automata (PDA) is one of the computational models based on the generalization of the Pushdown Automata (PDA). It's one of the theoretical computational models that utilizes an additional stack, which extends the capabilities of general one-way one-stack pushdown automata. By definition, a general two-stack PDA is a specific type and is a subset of general multistack or k-stack machines. A k-stack machine's input is obtained from an input source the same way as that of a PDA, rather than placing the input on a stack or tape, as the Turing Machine does. A multistack machine also has a finite control and a finite set of states. In terms of movement, both the moves of multistack machines and two-stack PDA are based on the state of the finite control, the input symbol read from the finite input alphabet, and the stack symbol of the stacks. A general k-stack machine can accommodate ε inputs. However, to make a k-stack machine deterministic (including two-stack PDA), there must not be a choice between an ε input and a non-ε input in any situation. When a move is made, a k-stack machine (along with two-stack PDAs) can also change to a new state and replace the top symbol of the stack/s with a string of zero or more stack symbols.[1]

To formally define two-stack deterministic pushdown automata, we extend Hopcroft's[1] formal definition of a one-stack PDA and San Pietro's[3] formal definition of a two-stack deterministic PDA by taking into account the second stack and its operations. By virtue of the extension, a Two-Stack Deterministic Pushdown Accepter (PDA) is defined as a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_I, \delta, F)$ where:

- Q - finite set of states.
- Σ - finite input alphabet.
- Γ - finite stack alphabet.
- δ -Q x Σ^* x Γ^* x $\Gamma^* \rightarrow Q$ x Γ^* x Γ^* is the transition function based on the current state, read input, and top stack symbols of both stacks.
    - Such that:
        - δ(current state, read input, pop from stack 1, pop from stack 2) $\rightarrow$ δ(next state, push into stack 1, push into stack 2).
        - If δ(q, ε, X, Y) is defined where X & Y can be the same symbol and X & Y ε Γ and a ε Σ, δ(q, a, X, Y) is undefined. The same concept is applied ifδ(q, a, X, Y) is defined such that δ(q, ε, X, Y) must be undefined.
- q_I - initial state, q_I ε Q.
- Z - initial stack symbols of both stacks, such that Z_a & Z_b are the initial symbols of the first and second stack respectively and Z is a proper subset of Γ and Z_a & Z_b ε Z It is also possible that both stacks have the same initial stack symbol.
- F - The set of accepting or final states such that F is a proper subset of Q.

It is important to note that a two-stack deterministic PDA (or general pushdown automata) has two ways to signal string acceptance: accepting a string by final state or empty stacks. In addition, a two-stack deterministic PDA can also have an empty set of accepting or final states, signifying empty stacks as basis for string acceptance. For the first condition, a string

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

can be accepted if the machine enters a final state while the whole string is processed. The second condition is if the machine is able to process the whole input string with the stacks empty. In a deterministic context, the languages accepted by one method may not be the same as those of the other method as per (Hopcroft et al., 2001).[1]

Furthermore, a two-stack deterministic PDA has the same total state as that of a two-stack non-deterministic PDA, such that their total state is the current state of the machine and the contents of both stacks. A two-stack PDA's total state is also crucial in highlighting deterministic and non-deterministic transitions of a given two-stack PDA (as per the given formal definition above).

The input format of a two-stack deterministic PDA is expounded on the formal definition above and differs slightly from that of a two-stack non-deterministic PDA in terms of valid transitions given a current total state. Moreover, its input string can be read or processed on either a single-character or multiple-character basis via a leftmost to rightmost sequence. To simplify, on each machine operation, a single or multiple characters of an input string may be read or processed each time, processing the input string from the leftmost to the rightmost part. Since general pushdown automata are accepters by nature or definition, they don't have any explicit output formats except indicators if a string is accepted or rejected.

In terms of machine operations, a two-stack deterministic PDA shares similarities to two-stack non-deterministic or regular PDAs in general, with an additional stack and the nature of determinism in the transitions taken into account for the machine operations. To further expound the machine operations based on the formal definition above, a two-stack deterministic PDA can perform the following operations:

- Set the initial stack symbols of both stacks.
- Read an input string's symbols that are elements of a finite input alphabet. A two-stack deterministic PDA can individually or collectively read an input string's symbols from left to right.
- Transition to a state. A two-stack deterministic PDA can transition to a new state or the current state depending on the current total state: the machine's current state and the contents of the two stacks. It is crucial to consider determinism in this context since the transition rules of a two-stack deterministic PDA slightly differ from those of two-stack non-deterministic PDAs such that they can also transition to more than one state given the same current total state.
- Stack manipulation. Two-stack deterministic PDAs can manipulate each or both of the stacks via pushing empty, one or more symbols into the stacks or popping their top symbols.

To better understand the machine operations, consider the following example of a two-stack deterministic PDA that recognizes the context-sensitive language $L = \{a^n b^n c^n \mid n \geq 1\}$

[!h]

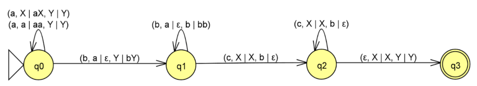


Figure 1.0: *A two-stack deterministic PDA accepting the same no. of a's, b's, and c's starting from a to c*

Although there are different formats of how the elements of the transitions are arranged, for the sake of consistency, we will follow the format throughout the whole paper: **(read input, pop from stack 1, push to stack 1, pop from stack 2, push to stack two)**. The given machine accepts strings that contain succeeding same numbers of a, b, and c with a condition that rejects empty inputs. . In addition, the direction of the arrows will indicate the next state. For instance, some accepted strings include abc, aabbcc, aaabbbccc, etc., while some rejected strings include bac, abbc, etc. To better understand how a two-stack deterministic PDA works, let's say we have an input string of: aabbcc. The trace would be the following:

Step	**Current State**	**Input Read**	**Action**	**Stack 1**	**Stack 2**
1	q0		Start	X	Y
2	q0	a	Read a, pop X, push aX, pop Y, push Y	aX	Y
3	q0	a	Read a, pop a, push aa, pop Y, push Y	aaX	Y
4	q1	b	Read b, pop a, push ε, pop Y, push bY	aX	bY
5	q1	b	Read b, pop a, push ε, pop b, push bb	X	bbY
6	q2	c	Read c, pop X, push X, pop b, push ε	X	bY
7	q2	c	Read c, pop X, push X, pop b, push ε	X	Y
8	qf	ε	Read ε, pop X, push X, pop Y, push Y **Accept String by Final State**	X	Y

Figure 1.1: *A Two-Stack Deterministic PDA's Trace Given a Valid Input String*

If we're going to process an input string (e.g. abbcc) that doesn't satisfy the language, the following would be the trace:

Step	**Current State**	**Input Read**	**Action**	**Stack 1**	**Stack 2**
1	q0		Start	X	Y
2	q0	a	Read a, pop X, push aX, pop Y, push Y	aX	Y
3	q1	b	Read b, pop a, push ε, pop Y, push bY	X	bY
4	q1	b	**Reject String**. Can't pop a from stack 1	X	bY

Figure 1.2: *A Two-Stack Deterministic PDA's Trace Given a Valid Input String*

In this trace, we can see that the machine rejected the string after the machine tried to read the second b of the string. It is due to the fact that the machine tried to pop an 'a' symbol in the first stack even if the symbol X is on the top.

Formal Language and Computational Power

In 1956, Noam Chomsky, the renowned American linguist and cognitive scientist, introduced the "Chomsky Hierarchy," a classification of formal grammars crucial to Formal Language Theory and Computer Science. The hierarchy consists of four levels: type-3 (Regular grammars) at the bottom, followed by type-2 (Context-Free grammars), type-1 (Context-Sensitive grammars), and finally, type-0 (Recursively Enumerable grammars) as the most powerful and general level.

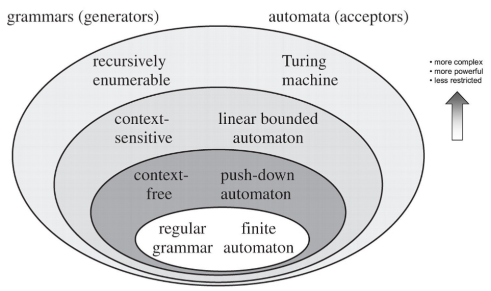


Figure 2.0: *Chomsky Hierarchy*

Two-Stack Deterministic Pushdown Automata & **Deterministic Finite Automata**

As per the Chomsky hierarchy, deterministic finite automata (which is a proper subset of non-deterministic finite automata) can recognize some regular grammars whereas two-stack deterministic pushdown automata can recognize more powerful languages including some context-sensitive and recursive languages (figure 1.0's language is both context-sensitive and recursive). Due to their difference in the hierarchy, we can prove that a deterministic finite automaton (DFA) can be converted into a two-stack deterministic PDA through the following steps:

- Maintain all the states (including initial and final states) from the DFA in the two-stack deterministic PDA.
- Maintain all the transitions from the original DFA and ensure that no stack manipulations can be performed with the transitions.
- Since two-stack deterministic PDAs have two ways to accept strings, if string acceptance is based on the final state, it doesn't matter if the stacks have initial stack symbols or not. If string acceptance is based on an empty stack, if stack symbols are initialized, ensure that a transition leading to a final state pops the symbols of both the stacks. If there are no initial stack symbols, proceed with enforcing all transitions to have no stack manipulations or operations (empty pop and push symbols).

To prove that a deterministic finite automaton can be converted into a two-stack deterministic PDA, we first design a DFA that recognizes the regular grammar language L = {w | w has both an even number of 0's and an even number of 1's}.

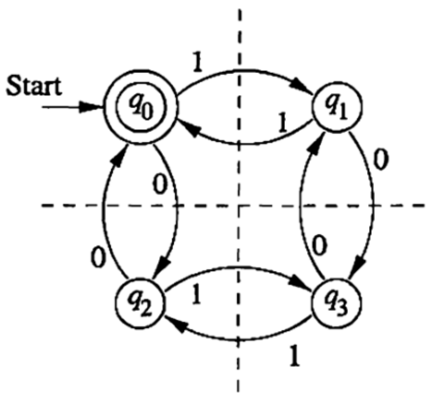


Figure 3.0: *Transition Diagram of the Language as per (Hopcroft et al., 2001)*

The equivalent two-stack deterministic PDA of the diagram in figure 3.0 is the following diagram:

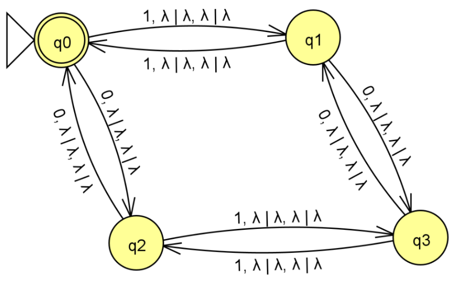


Figure 3.1: *Transition Diagram of the Language as per (Hopcroft et al., 2001) in Two-Stack Deterministic Form*

The nature of determinism remains in figure 3.1 due to the fact that from a given combination of a current state, current read input, and top stack symbols of both stacks (to be popped), there is only at most one state that the machine can transition to. In addition, all the transitions of each state are unique from each other: given a state, even if both the top stack symbols of both stacks are lambda, the machine can either read 1 and go to one state or read 0 and go to another state, enforcing only a single possible choice of transition for each state and read input.

However, not all two-stack deterministic PDAs can be converted into deterministic finite automata. Consider the previous example of a two-stack deterministic PDA (figure 1.0) that recognizes the context-sensitive language $L = \{a^n b^n c^n \mid n \geq 1\}$. DFAs can only recognize some regular grammars, but it cannot recognize languages more powerful than regular grammars as per the Chomsky hierarchy. To prove that the DFA can't recognize the language $L = \{a^n b^n c^n \mid n \geq 1\}$, we can use the Pumping Lemma for regular languages such that if the language is proven not a regular language via Pumping Lemma, we can conclude that the DFA can't recognize it.

We can prove that the language is not regular via proof by contradiction, where we assume that $L = \{a^n b^n c^n \mid n \geq 1\}$ is a regular language and there exists a pumping length p for the language such that the the length of the input string S $\geq p$ may be divided into 3 parts $S = xyz$ such that all the following conditions must be **true**:

- $xy^i z \,\varepsilon\, L$
- $|y| > 0$
- $|xy| \leq p$

Solution: Assume $L = \{a^n b^n c^n \mid n \geq 1\}$ is a regular language and the pumping length $= p$. S = $a^p b^p c^p$ $p = 3$ and S = $aaabbbccc$
We can check the first condition '$xy^i z\ \varepsilon\ L$ for every $i \geq 0$' and if it isn't satisfied, we can conclude that the language is indeed non-regular. For clarity, let the string be in the format of $x|y|z$. Consider the following case:
Case 1: The y part is among the a's
S $= a|aa|bbbccc$. If $i = 2$ in $xy^i z$, then $xy^2 z \rightarrow a|aaaa|bbbccc$. Since the p in this string's a^p is equal to 5, it is not equal to the original pumping length p, which is $p = 3$. The string S $=$ $aaaaabbbccc$ is also not an element of the language, therefore, the first condition is not satisfied.

We can still proceed to succeeding cases such that the y part is among the a's and b's, b's, b's and c's, and c's. However, one case that doesn't satisfy the first condition is enough to prove that the language is not regular. We can further conclude that a deterministic finite automaton does not recognize the language.

By proof by construction, we can also conclude that two-stack deterministic PDAs are more powerful than DFAs.

Two-Stack Deterministic Pushdown Automata & Two-Stack Non-Deterministic Pushdown Automata As per the theorem 8.13 of (Hopcroft et al, 2001), If a language L is accepted by a Turing machine, then L is accepted by a two-stack machine, including a two-stack non-deterministic PDA. It was proven via the simulation of a Turing machine with two stacks. Since the focus of this case study is mainly on two-stack deterministic PDA, we will not expound on the proof. Furthermore, as per the Chomsky hierarchy, if Turing machines can recognize recursively enumerable languages, then a two-stack non-deterministic PDA can also recognize recursively enumerable languages. However, since two-stack deterministic PDA are a proper subset of two-stack non-deterministic PDA, there are languages that can be recognized by two-stack non-deterministic PDA that can't be recognized by two-stack deterministic PDA. Consider the following non-deterministic context-free language:

$L = \{ww^R \mid \text{w}\ \varepsilon\ \{0,1\}^*\}$

We can prove that both one-stack and two-stack non-deterministic PDA can recognize the language above by creating a one-stack non-deterministic PDA diagram that recognizes the language, and converting that diagram into a two-stack non-deterministic PDA. The following figures show the results of the creation and conversion:

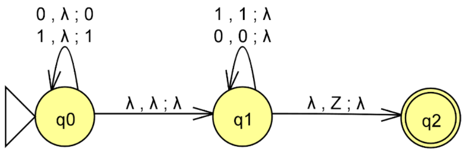


Figure 4.0: *Transition Diagram of the Language* $L = \{ww^R \mid \text{w}\ \varepsilon\ \{0,1\}^*\}$

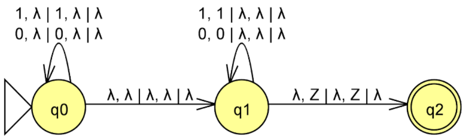


Figure 4.1: *Transition Diagram of the Language* $L = \{ww^R \mid \text{w}\ \varepsilon\ \{0,1\}^*\}$ in Two-Stack Non-Deterministic Form

The concept of converting a one-stack non-deterministic PDA to a two-stack non-deterministic PDA is the same as that of converting DFA into two-stack deterministic PDA: maintain all the states, maintain all the transitions, and ensure that the second stack is never used or manipulated. If both stacks have initial symbols, ensure that no symbols will be pushed into the second stack and the initial symbol of the second stack will be popped at some point (if string acceptance is via empty-stack basis).

However, even a two-stack deterministic PDA will not be able to recognize the language $L = \{ww^R \mid \text{w}\ \varepsilon\ \{0,1\}^*\}$. The reason is because a two-stack deterministic PDA will not be able to identify the division between the original string and the reversed string. The two-stack deterministic PDA will not be able to identify whether the read 0's and 1's belong to the original string or the reversed string. A non-deterministic approach is required such that for every character read from the input string(0 or 1 for instance), there exists another timeline/s or input trace that non-deterministically divide/s the string into two parts: original and reversed string. If the string is divided unevenly, the timeline will eventually lead to a rejection, else, if such a timeline exists wherein the string is divided evenly, the next approach of the machine is to deterministically check if the input character read in the reversed string part matches the top of the stack. If they match, the machine pops from the stack the symbol read on the second part of the input string, ensuring that the only way the whole input string can be read is if the second part of the string is indeed the reverse of the first part.

However, if we modify the previous language example and add a clear indicator of the division of the string such that $L = \{wxw^R \mid \text{w}\ \varepsilon\ \{0,1\}^*\}$ where the original w and reversed w are divided by the symbol x, then this becomes deterministic since we know that the only way we can reach the second part of the input string (reverse of w) is if we deterministically read an x symbol from the input string. The following is an example of a two-stack deterministic PDA that recognizes the language. $L = \{wxw^R \mid \text{w}\ \varepsilon\ \{0,1\}^*\}$:

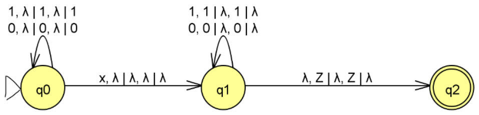
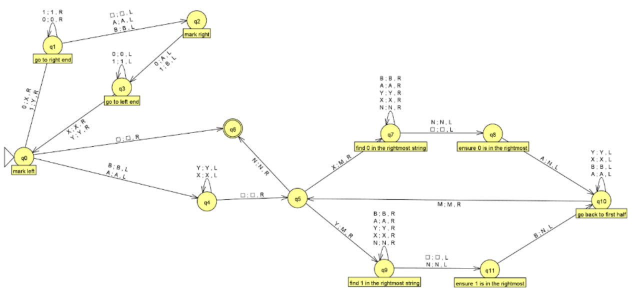


Figure 4.2: *Transition Diagram of the Language* $L = \{wxw^R \mid \text{w}\ \varepsilon\ \{0,1\}^*\}$ in Two-Stack Deterministic Form

By virtue of proof by construction and counter-examples, we can conclude that two-stack deterministic PDA are weaker in terms of computational power than two-stack non-deterministic PDA.

Two-Stack Deterministic Pushdown Automata & Turing Machines In addition to the theorem proved by Hopcroft et al. such that any language L accepted by a Turing Machine is also accepted by a two-stack machine, a study by Vladik & Olga went into depth of the similarities of the Two-stack PDA and the Turing Machine. In summary of their work, It is revealed that the Two-stack PDA and the Turing Machine are equivalent, but they aren't equally efficient in solving such specific problems. Turing machines are considered the more practical and preferred model for expressing general computations and algorithms due to their simplicity and ease of understanding. On the other hand, Two-Stack PDAs are useful in theoretical discussions, language recognition, and in proving properties related to the computational complexity of certain languages. [2]

However, as per our previous proof, a two-stack deterministic PDA can't recognize the non-deterministic context-free language $L = \{ww^R \mid w\ \varepsilon\ \{0,1\}^*\}$. This was also proven by San Pietro [3] in 1992 via the Ogden Lemma. To prove if a Turing Machine is stronger than a two-stack deterministic PDA, we can create a Turing Machine that recognizes the given language.



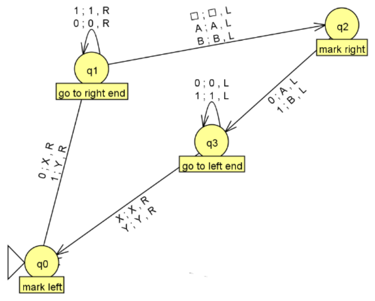
go to right end
mark right
go to left end
mark left
find 0 in the rightmost string
ensure 0 is in the rightmost
go back to first half
find 1 in the rightmost string
ensure 1 is in the rightmost

Figure 5.0: *Transition Diagram of the Language* $L = \{ww^R \mid w\ \varepsilon\ \{0,1\}^*\}$ in a Non-Deterministic Turing Machine form

The general idea of the diagram above is for the first part of the algorithm, it needs to divide the string into two parts: w and its reverse. For initialization, the following part of the diagram ensures that the first part of the string has 0's and 1's that correspond to X and Y respectively while the second part of the string has 0's and 1's that correspond to A and B respectively.

Figure 5.1: *Part of the Diagram Responsible for Deterministic Division of an Input String into w and* w^R

This is the first part of the algorithm. Everytime a 0 or a 1 is read from the left and is converted into X or Y, the Turing machine goes to the right-end of the tape until the blank symbol, letter A, or letter B symbol is encountered. Afterwards, if a 0 or 1 is read from the right side and is converted into A or B, the machine goes back to the left-end of the input string until an X or Y symbol is encountered. Let's say we have an input of "001100", then the resulting string on the tape is "XXYBAA".



Afterwards, the algorithm proceeds by forcing the tape head to move the left-end as shown by a part of the diagram presented below:

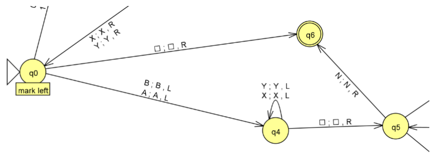


Figure 5.2: *Part of the Diagram Responsible for Moving the Tape Head to the Left-End*

After the tape head moves into the left-end, the machine then commences the algorithm responsible for checking the validity of the input string. If an X (which is equivalent to 0) is read and converted to M, the tape head moves to the right-end and checks if the cell contains a letter A (which is equivalent to 0) and replaces it with N, if a Y (which is equivalent to 1) is read, the tape head moves to the right-end and checks if the cell contains a letter B (which is equivalent to 1) and replaces it with N. After the machine checks the corresponding value in the right-end of the tape, it goes back to the left-end of the tape until a letter M is encountered. Afterwards, the machine reads 0 or 1 again and the same process is repeated. The idea of this algorithm is that it checks if the leftmost number on the string w is the same with the rightmost number on the string w^R to ensure that the string corresponds to the language. The same goes with the 2nd number from the leftmost of w and the 2nd number from the rightmost of w^R and the succeeding nth numbers of both ends until no more comparisons are possible. The moment the machine detects that the two nth number from the leftmost and rightmost are not the same, the machine automatically rejects the string. The following diagram shows what part of the machine is responsible for the algorithm.

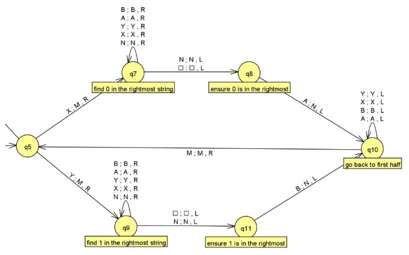


Figure 5.3: *Part of the Diagram Responsible for Checking if the Input String Corresponds to the Language*

Since we have created a Turing Machine that recognizes the language $L = \{ww^R \mid w\ \varepsilon\ \{0,1\}^*\}$, we can therefore conclude that Turing Machines are more powerful than deterministic two-stack PDA in terms of computational power. To summarize, two-stack deterministic PDAs are stronger than deterministic finite automata. However, they are weaker than both two-stack non-deterministic PDA and Turing machines. In terms of the Chomsky hierarchy, two-stack deterministic PDAs fall somewhere between recursively enumerable (type-0) and context-sensitive (type-1) languages. It is also important to note that two-stack deterministic PDAs can't always recognize context-sensitive, context-free, and regular languages due to its nature of determinism.

Applications

Computational models play a crucial role in tackling diverse problem sets, proving to be highly valuable in various domains. However some models exhibit a wider range of applications compared to others, such as the two-stack deterministic pushdown automaton, which can solve a number of problems. This model is a powerful and versatile tool for exploring the limits and capabilities of algorithms and languages. With its two stacks, it can address a wide range of computational problems, making it a valuable resource in theoretical computer science. This added complexity allows the Two-Stack PDA to tackle more intricate languages and grammars. For example, we can leverage the power of regular expressions (regex) in conjunction with a two-stack pushdown automaton to validate the format of an email address. By employing regex, we can define specific grammar rules that the email must adhere to, ensuring that it follows the required pattern of "username@domain.com" or it can also accommodate the pattern "username@domain.edu.ph." For this example we could consider the the grammar rules:

- C - represents character between a - z or A - Z
- I - a number between 0 - 9
- SC - any special character that does not involve the special characters "@" and "."
- @ - the literal special character "@"
- "." - the literal special character "."

This grammar rule represents all possible finite input alphabets that will be stored in stack 1 and stack 2. The first stack in the automaton is responsible for validating the characters comprising the username and domain, verifying that they consist of alphanumeric characters, dashes, and underscores. Meanwhile, the second stack is utilized to confirm the presence of the essential '@' and "." symbol in the email address. To signal for string acceptance is through accepting a string by final state and not by empty stack. Using that logic, here are the possible sample input and output to showcase what would be accepted and what would be rejected.

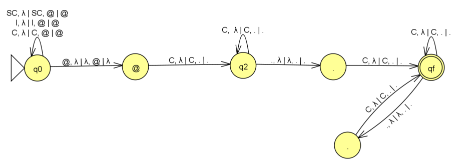


Figure 6.0: *A Two-Stack Deterministic PDA Simulating Email Validation*

Input: CCCCSCICCSC@CCCCC.CCC
Output: Accepted

Input: IIICSCCCCSC@CCCCCC
Output: Rejected

Input: CCSCSCCCCSC@CCCCC.CCC.CC
Output: Accepted

Through this approach, the two-stack pushdown automaton can efficiently process the input email and validate its format according to the predetermined grammar rules. This combination of regex and a two-stack pushdown automaton provides an insightful demonstration of how automata theory and pattern matching can be effectively applied to practical tasks like email validation.

REFERENCES

[1] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. 2001. *Introduction to automata theory, languages, and computation*. Vol. 32. ACM New York, NY, USA. 60–65 pages.

[2] Vladik Kreinovich and Olga Kosheleva. 2018. A Turing Machine Is Just a Finite Automaton with Two Stacks: A Comment on Teaching Theory of Computation. (2018).

[3] Pierluigi San Pietro. 1992. TWO-STACK AUTOMATA. (1992).