STALCGM Case Study about Deterministic Pushdown Automata
Term 3 AY 2022 - 2023

Samuel Vincent Cheng
2401 Taft Avenue
Manila, Philippines
samuel_vincent_cheng@dlsu.edu.ph

Tyrone Angelo Uy
2401 Taft Avenue
Manila, Philippines
tyrone_uy@dlsu.edu.ph

Kendrick Mikhael Pua
2401 Taft Avenue
Manila, Philippines
kendrick_mikhael_pua@dlsu.edu.ph

ABSTRACT
This paper presents a case study on deterministic pushdown automata (DPDAs), and an implementation of them through a web application. The DPDA model is nearly identical to that of the PDA, with the added constraint that the machine must be deterministic. This constraint limits the set of languages acceptable by the model to a proper subset of context-free languages, meaning the DPDA is strictly weaker than the PDA. Despite being weaker than the PDA, the stack ensures that DPDAs are still more powerful than finite state accepters. The paper also explains one application of the model in computer language parsing.

Author Keywords
automata theory, deterministic pushdown automaton, formal languages, theory of computation

CCS Concepts
•**Automata Theory → Compiler Theory;**

Machine Definition
DPDAs are accepters similar to deterministic finite accepters (DFAs) in the sense that inputs are stored in a have a read-only, one-way input tape, and that they are deterministic, meaning at any given moment, there is at most 1 valid transition for the machine to take. Unlike DFAs, they allow transitions on the empty string, have a stack, and support the operations of pushing symbols to and popping symbols from the stack.

The formal definitions of PDAs and DPDAs were adapted from the textbook titled "Principles of Compilers: A New Approach to Compilers Including the Algebraic Method" [4].

A pushdown automaton (PDA) is defined as a 7-tuple M = (Q, Σ, Γ, δ, q_0, Z, F) where:

Q - is a finite set of states

Σ - is a finite input alphabet,
Γ - is a finite stack alphabet,

The transition function δ: $Q \times (\Sigma \cup \{\lambda\}) \rightarrow$ finite subsets of $Q \times \Gamma$

q_0 - is the initial state, $q_0 \in Q$
Z - initial stack symbol, $Z \in \Gamma$
F - is the set of final states, $F \subseteq Q$

A string is accepted by a PDA if, after reading the whole string, the PDA is in a final state and the stack is empty.

Since the book did not provide a definition for the overall state of a PDA, we instead based the definition from the textbook "Machines, languages, and computation" by Denning, et. al. [3]. The overall state of a PDA is a 2-tuple $(q, \phi) \in Q \times \Gamma^*$ where q is the state the machine is currently in and ϕ is the contents of the stack.

A deterministic pushdown automaton (DPDA) is a PDA M = (Q, Σ, Γ, δ, q_0, Z, F), such that:

$$\forall q \in Q, a \in \Sigma \cup \lambda, b \in \Gamma \cup \lambda \left(|\delta(q,a,b)| \leq 1 \wedge (\delta(q,\lambda,b) \neq \emptyset \rightarrow \forall x \in \Gamma(\delta(q,\lambda,x) = \emptyset))\right)$$

That is, A DPDA is a PDA whose transition function must adhere to the following 2 constraints:
1. For every state q, if there is a lambda transition from q that pops some stack symbol b, there are no other transitions from q that pop b, and if there is a lambda transition from q that pops nothing, it is the only transition from q that pops nothing.
2. For every state q, stimulus symbol a, and stack symbol b, there is at most one transition from q on a that pops b.

Additionally, the set of languages recognized by some DPDA is known as the deterministic context-free languages.

Example Machine
Let us define a DPDA M = (Q, Σ, Γ, δ, q_0, Z, F) that recognizes the language $L = \{\omega \in \{0,1\}^* \mid \omega = 0^n 1^n$ for some $n \geq 1\}$ where:

Q ={A,B,C}
Σ ={0,1}
Γ ={X,Z}

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

δ(A,0,λ) = (A,X)
δ(A,1,X) = (B,λ)
δ(B,1,X) = (B,λ)
δ(B,λ,Z) = (C,λ)
q_0 = A
F = {C}

When the input string 01 is fed to the machine, the machines actions are as follows: At time = 0 time units, overall state of M is (A, Z). At time = 1, M reads a 0, going to state A, popping nothing and pushing X. Overall state is (A, XZ). At time = 2, M reads a 1, going to state B, popping X and pushing nothing. Overall state is (B, Z). At time = 3, M takes the lambda transition, going to state C, popping Z and pushing nothing. Overall state is (C, λ). Since the machine ended in a final state and the stack was empty, the string is accepted.

Formal Language and Computational Power

This portion discusses the computational power of the DPDA relative to the models discussed in class (FSAs, PDAs, Turing Machines) and shows where the model lies in the Chomsky Hierarchy.

DPDAs and Finite-State Accepters

Any DFA can be converted to a DPDA. Since DFAs and non-deterministic finite accepters (NFAs) are Turing equivalent, DPDAs are at least as powerful as FSAs.

There is a lack of credible sources that explain how DFAs can be converted to DPDAs, possibly due to slight differences on how PDAs (and by extension, DPDAs) are defined in the literature. Fortunately, an algorithm was easy to formulate ourselves.

[4] defines a DFA as M=(Q,Σ,δ,q_0,F_1) where:

Q - is a finite set of states
Σ - is a finite input alphabet
The transition function : Q x S $\rightarrow$ Q
q_0 - is the initial state, $q_0 \in$ Q
F_1 - is the set of final states, $F_1 \subseteq$ Q

Given a DFA M1=(Q_1,Σ_1,δ_1, q_{0_1}, q_F), the equivalent DPDA is the machine M2=(Q_2,Σ_2,Γ,δ_2,q_{0_2},Z,F_2) where:

$Q_2 = Q_1 \cup F_2$
$\Sigma_2 = \Sigma_1$
Γ = {Z}
δ_2 = (q,a,b) = {

{(h, λ)}	if q $\in F_1 \wedge$ a = $\lambda \wedge$ b = Z
{(q',λ)}	if δ_1(q,a)= q' $\wedge$ b = λ
$\emptyset$	otherwise

}
$q_{0_2} = q_{0_1}$
F_2 = {h} where $h \notin Q_1$

Every transition in M1 has a corresponding transition in M2 on the same input symbol that pops nothing, pushes nothing, and has the same source and destination states. This allows M2 to simulate M1 because none of those transitions ever perform any operation on the stack. The stack is never utilized except for storing the initial stack symbol at the start and possibly popping it at the end.

M2 has every state M1 does, plus one more. To account for the initial stack symbol, M2 has a single final state *h*, that all of M1's final states (which are no longer final in M2) transition to on the empty string while popping Z and pushing nothing. This means that M2 has $|F_1|$ more transitions than M1 has.

Since any state in M2 will have, at most, 1 transition on the empty string, and this transition will pop *Z*, and all transitions in the machine that are not on the empty string pop nothing, the 1st criteria for determinism is satisfied by M2. Since the transitions in M1 did not cause any non-determinism, their corresponding transitions in M2 will not either, and since all the transitions on the empty string in M2 do not cause non-determinism either, then the 2nd criteria for determinism is also satisfied by M2.

DPDAs Do Not Always Have Equivalent DFAs

Though every FSA has an equivalent DPDA, not every DPDA has an equivalent FSA. This can be proven easily by finding a language that is deterministic context-free and not regular.

Take the aforementioned language L accepted by the example machine defined in the "Machine Definition" section. We can prove that there is no FSA that recognizes L using pumping lemma.

Pumping lemma states that if a language is regular, then there exists some positive integer p such that every string *s* in that language of length p or longer can be expressed as *s* = xyz with 3 conditions being satisfied [3]:

1. |xy| $\leq$ p
2. |y| > 0
3. xy*z is in the language

For any string in L, there are 3 possible forms y can take that do not violate the 2nd requirement. We will now show that none of these forms can satisfy the 3rd condition, that xy*z $\in$ L.

Case 1: y $\in$ 00*
Any string in L 0^n1^n that has 1 or more of its 0s pumped would become $0^{n+k}1^n$ where k is a constant positive integer. Since n + k $\neq$ n, this string is not in L.

Case 2: y $\in$ 11*
Similarly to case 1, Any string in L 0^n1^n that has 1 or more of its 1s pumped would become 0^n1^{n+k} where k is a positive constant integer, and since n + k $\neq$ n, this string is not in L.

Case 3: y $\in$ 00*11*
If y is pumped 1 or more times, the resulting string s contains yy as a substring. Since y ends with a 1 and starts with a 0, then 010 is substring of s, meaning s $\notin$ L

From this, we can conclude that for any s $\in$ L, there is no way to express s as xyz such that pumping lemma is satisfied. Therefore, by Modus Tollens, L is not regular. Since L is not

regular, there is no FSA that recognizes it. Since L can be recognized by a DPDA but not by any FSA and DPDAs are at least as powerful as FSAs, we conclude that DPDAs are more powerful than FSAs.

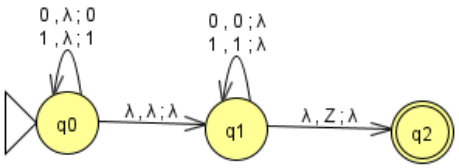
Deterministic Pushdown Automaton and (Non-Deterministic) Pushdown Automaton

Unlike FSAs, the power of the PDA model is lessened with the addition of determinism. There are 2 steps to proving this. First we prove that PDAs are at least as powerful as DPDAs. Since, by definition, every DPDA is a PDA, every DPDA certainly has an equivalent PDA. Second, we prove that DPDAs are less powerful than PDAs. It is known that the set of deterministic context-free languages is a proper subset of the context-free languages [4], the set of all languages recognizable by a PDA. Since PDAs are at least as powerful as DPDAs and there exist languages recognizable by some PDA but not by any DPDA, we conclude that PDAs are more powerful than DPDAs.

Languages that are recognizable by PDAs but not DPDAs necessitate making decisions that cannot be made deterministically. There are context-free languages for which a DPDA lacks the power to recognize, as it cannot make "choices" or "guesses" while computing something. In contrast, an NPDA can non-deterministically explore different paths, enabling it to recognize a broader set of languages.

An example is the language of even length palindromes on $\{0,1\}$, $L = \{ww^R | w \in \{0,1\}^*\}$.

This NPDA recognizes L:



The problem demands both halves of the string be compared, meaning the machine must somehow identify where 1 half ends and the other begins. The NPDA given above does so nondeterministically. The nondeterminism in this machine arises from the transitions $\delta(q_0,\lambda,\lambda) = \{(q_1,\lambda)\}$, $\delta(q_0,0,\lambda) = \{(q_0,0)\}$ and $\delta(q_0,1,\lambda)=\{(q_1,1)\}$. At any point when the machine is at q_0 and reads an input symbol, there will always be 2 legal transitions to take. In the situation where there are multiple transitions that can be taken, it is as if multiple copies of the machine are spawned, each taking a different transition. If any one of those copies accepts the string, then the machine as a whole accepts the string. This way, the machine eventually finds the correct halfway point by trying every possibility.

No equivalent DPDA exists for this because there is no way for this decision to be made deterministically with a PDA. The DPDA lacks the means to differentiate between the 2 halves. This renders the recognition of even length palindromes impossible by a DPDA.

The Relative Power of the DPDA

Given that we now know DPDAs are strictly weaker than PDAs but stronger than FSAs, we can now determine where the model lies in the Chomsky hierarchy.

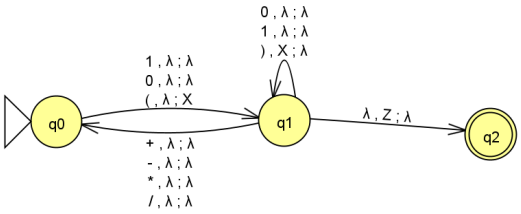
Chomsky Hierarchy of Abstract Machines and Languages Including DPDAs:

Automaton	**Language**
Turing Machine	Recursively Enumerable Languages
Linear Bounded Automata	Context-Sensitive Languages
Pushdown Automaton	Context-Free Languages
Deterministic Pushdown Automaton	Deterministic Context-Free Languages
Finite State Automata	Regular Languages

Applications

One application of the concept of Deterministic Pushdown Automata can be found in language parsing, specifically, computer language parsers. Parsers are utilized to, given a certain grammar, determine whether a particular string is generated by said grammar. An example of a parser using the DPDA abstract model is Simple LR Parsing (SLR(1) parsing). Simple LR Parsing utilizes a bottom-up, non-recursive strategy to parse strings for a given grammar. This method of parsing has been compared with other parsing methods, and tested and used in machines like the PDP-10 [2].

Practical Application: Simple Arithmetic Parser



A common task DPDAs accomplish in parsing is the parsing of arithmetic expressions in computer languages. The figure above shows one such model. Something interesting about this model is that it recognizes arithmetic expressions involving a juxtaposition of 2 expressions/values (e.g. (10)(100)). This expression may be considered invalid by some programming languages, but some, such as the Julia programming language, treat these strings as valid, and interprets them similarly to how they are in mathematics [1].

REFERENCES

[1] 2023. Intergers and FLoating-Point Numbers. (2023). https://docs.julialang.org/en/v1/manual/integers-and-floating-point-numbers/#man-numeric-literal-coefficients

[2] Jacques Cohen and Martin S. Roth. 1978. Analyses of Deterministic Parsing Algorithms. *Commun. ACM* 21, 6 (jun 1978), 448–458. DOI: http://dx.doi.org/10.1145/359511.359517

[3] Peter J Denning, Jack B Dennis, and Joseph E Qualitz. 1980. Machines, languages, and computation. (1980).

[4] Yunlin Su and Song Y Yan. 2023. Principles of Compilers. *SpringerLink* (2023). DOI: http://dx.doi.org/10.1007-978-3-642-20835-5