Electronic Colloquium on Computational Complexity, Report No. 81 (2001)

Pushdown Automaton with the Ability to Flip its Stack

Palash Sarkar*
Centre for Applied Cryptographic Research
Department of Combinatorics and Optimization
University of Waterloo
200 University Avenue West
Waterloo, Ontario
Canada N2L 3G1
e-mail: psarkar@cacr.math.uwaterloo.ca

Abstract

We introduce the idea of pushdown automata with the ability to flip its stack. By bounding the number of times the stack can be flipped we obtain a hierarchy of language classes from the context free languages to the recursively enumerable languages. We show that each class in the hierarchy is nonempty and conjecture that the hierarchy is strict. Also we provide some interesting open problems for this new kind of automata.

Keywords : theory of computation, formal languages, push down automata.

1 Introduction

A pushdown automata is a finite automata equipped with an auxilliary storage device in the form of a stack. We provide additional power to the automata by allowing it to flip the stack. This appears to be a very natural and intuitive operation. Allowing the stack to be flipped actually allows the automata to read and write at both ends of the stack. Thus the auxilliary storage device becomes a double ended queue (a dequeue) (see [2, 3] for dequeue machines). It is clear that a dequeue can simulate two stacks and hence the computational model becomes equivalent to the Turing machine model.

Here we consider the number of times the stack is flipped to be a resource. This seems to be consistent with the basic intuition of a resource, since a flip of the stack can have a cost factor associated with it. We study the classes of languages that arise when this resource, i.e., the number

*On leave from the Indian Statistical Institute, Calcutta, INDIA. e-mail: palash@isical.ac.in

ISSN 1433-8092

of flips, is bounded. If the number of flips is zero, then the automata accepts the context free languages. On the other hand if the number of flips is unrestricted, then the automata accepts the recursively enumerable languages. Thus by bounding the number of flips we can get a hierarchy of language classes from the context free to the recursively enumerable. We identify this hierarchy and show that each class in the hierarchy is nonempty. It seems intuitive that the hierarchy is strict and we also obtain the languages that could perhaps be used to separate the classes in the hierarchy. However, we have not been able to actually prove that the hierarchy is strict.

2 Formal Model

The formal computational model is a flip-stack pushdown automata (FPDA) defined as follows. An FPDA $\mathcal{F} = (\mathcal{P}, \Delta)$, where $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ is a pushdown automata (PDA) and $\Delta : Q \to 2^Q$. If $\Delta(q) \neq \emptyset$ then the FPDA while in state q can flip its stack and go to any state in $\Delta(q)$.

We will follow Hopcroft and Ullman [1] for our notation and definition of PDA. However, we will differ from Hopcroft and Ullman in representing the stack contents as strings. We will assume that the stack bottom is on the left and the stack top is on the right.

The set of instantaneous descriptions (ID's) of $\mathcal{F}$ is equal to the set of ID's of $\mathcal{P}$. The transition between ID's of $\mathcal{A}$ are defined as follows.

$$(q, ax, \alpha Z) \underset{\mathcal{F}}{\vdash} (p, x, \beta\alpha) \text{ if } (q, ax, \alpha Z) \underset{\mathcal{P}}{\vdash} (p, x, \beta\alpha).$$

These are the transition of the underlying PDA. In addition, the following transitions are allowed, which allow the FPDA to flip its stack.

$$(q, x, Z_0\alpha) \underset{\mathcal{F}}{\vdash} (p, x, Z_0\alpha^r) \text{ if } p \in \Delta(q).$$
$$(q, x, Z\alpha) \underset{\mathcal{F}}{\vdash} (p, x, \alpha^r Z) \text{ if } p \in \Delta(q) \text{ and } Z \neq Z_0.$$

We do not want the FPDA to move the bottom of stack marker when the stack is being reversed. This is taken care of by the first kind of transition. Transitions of the above kind will be called stack reversals or stack flips.

Given an FPDA $\mathcal{F}$ and integer $k \geq 0$, we define the language $L(\mathcal{F}(k))$ in the following manner. For any string $x \in \Sigma^*$, $x \in L(\mathcal{F}(k))$ iff $(q_0, x, Z_0) \underset{\mathcal{F}}{\vdash} (p, \epsilon, \epsilon)$ for some $p \in Q$, and the number of stack flips is exactly k. Thus acceptance is by the empty stack. It is also possible to define acceptance by the final state. The language $L(\mathcal{F}(k))$ is the language $\mathcal{F}$ accepts using exactly k stack flips.

3 A Hierarchy of Languages

We next define language classes as follows. For each $k \geq 0$, we define

$$\mathcal{L}(k) = \{L : L = L(\mathcal{F}(k)), \text{ for some FPDA } \mathcal{F}\}. \tag{1}$$

Also the class obtained by an unrestricted number of flips is as follows.

$$\mathcal{L}(\infty) = \{L : L = L(\mathcal{F}),\ \text{for some FPDA } \mathcal{F}\}. \tag{2}$$

The next result is easy to see.

Proposition 3.1 *The following holds.*

1. $\mathcal{L}(0) = \mathbf{CFL}$.
2. $\mathcal{L}(\infty) = \mathbf{r.e.}$.

Proof : (1) If the FPDA is not allowed to make any stack reversal, then it clearly behaves like an ordinary PDA and hence can accept only the context free languages.
(2) Given a two stack machine $\mathcal{M}$, it is not difficult to construct an FPDA $\mathcal{F}$ with unrestricted number of flips that simulates $\mathcal{M}$. The essential idea is to allow the two stacks to grow at the top and at the bottom. The machine $\mathcal{M}$ updates both the stack simultaneously. Thus for each movement of $\mathcal{M}$, the FPDA $\mathcal{F}$ will require one stack reversal. It updates the stack which is on top, reverses the stack and updates the stack on the bottom. The reverse simulation, i.e., simulating an FPDA by a two stack machine is similar. ■

The following result shows the hierarchy of the language classes.

Theorem 3.1

$$\mathbf{CFL} = \mathcal{L}(0) \subseteq \mathcal{L}(1) \subseteq \mathcal{L}(2) \subseteq \ldots \subseteq \mathcal{L}\infty = \mathbf{r.e.}. \tag{3}$$

Proof : We have to show that $\mathcal{L}(k) \subseteq \mathcal{L}(k+1)$. Let $L \in \mathcal{L}(k)$. Then $L = L(\mathcal{F}(k))$, for some FPDA $\mathcal{F} = (\mathcal{P}, \Delta)$. We construct an FPDA $\mathcal{F}' = (\mathcal{P}', \Delta')$, such that $L(\mathcal{F}'(k+1)) = L$. Let q_0 be the initial state of $\mathcal{P}$. The PDA $\mathcal{P}'$ is a simple modification of $\mathcal{P}$ where the initial state of $\mathcal{P}$ is replaced by a new state q_0'. We do not change the function δ of $\mathcal{P}$. Thus $\mathcal{P}'$ by itself cannot move out from its initial state. However, we allow the FPDA $\mathcal{F}'$ to move from the initial state by setting $\Delta'(q_0') = \{q_0\}$ and for any state q of $\mathcal{P}$ we set $\Delta'(q) = \Delta(q)$.

In state q_0' the only possible move by the FPDA $\mathcal{F}'$ is a useless reversal of the stack which takes it to state q_0. Henceforth $\mathcal{F}'$ simulates $\mathcal{F}$ and hence $\mathcal{F}'$ accepts a string x iff $\mathcal{F}$ accepts x. Thus $L = L(\mathcal{F}'(k+1))$. ■

The immediate question is whether the hierarchy in (3) is strict. The following result settles the base case.

Proposition 3.2 *The language class $\mathcal{L}(0)$ is properly contained in the language class $\mathcal{L}(1)$.*

Proof : Consider the language $L = \{ww : w \in \{0,1\}^*\}$. It is well known that L is not a CFL. We show $L \in \mathcal{L}(1)$. Let $\mathcal{P}$ be a PDA that accepts the language $\{ww^r : w \in \{0,1\}^*\}$. We assume that $\mathcal{P}$ operates in two phases. In the first phase, $\mathcal{P}$ is in a state q_1 (the initial state) and pushes the input symbols to the stack. In the second phase, $\mathcal{P}$ is in a state q_2 and pops the top of stack if it matches the next input symbol. The change from state q_1 to q_2 is done nondeterministically. We construct a PDA $\mathcal{P}'$ which is essentially the same as the PDA $\mathcal{P}$ except that we remove all rules which allow $\mathcal{P}$ to move from state q_1 to state q_2. Thus by itself $\mathcal{P}'$ cannot move out from the state q_1. We define an FPDA $\mathcal{F} = (\mathcal{P}', \Delta)$ by setting $\Delta(q_1) = \{q_2\}$ and $\Delta(q_2) = \{q_2\}$.

We first claim that if x is accepted by $\mathcal{F}$ then x must be of the form ww. Since x is accepted, there must have been a transition from q_1 to q_2. Thus the stack must have been reversed at some point. Let the portion of input that was read before the stack was reversed be x_1 and the remaining input be x_2. Then $x = x_1x_2$. In state q_1 the FPDA pushes the input symbols onto the stack. Thus when the input x_1 has been read the stack also contains x_1. After reversal the stack becomes x_1^r. Since there has been exactly one stack reversal, in further computation the string x_2 is read and matched to the stack content x_1^r. Note that the matching is successful if the rightmost symbol of x_1^r matches the first symbol of x_2 and so on. Thus the FPDA accepts iff $x_2 = (x_1^r)^r = x_1$. But this means $x = x_1x_1$.

Conversely, it is not difficult to see that given any string of the form ww, the FPDA $\mathcal{F}$ accepts it. Thus $L \in \mathcal{L}(1)$. ∎

We now extend the idea in the proof of Proposition 3.2 to show that each language class in hierarchy (3) is nonempty. For each k, we define a language L_k as follows.

$$L_0 = \{x_1x_1^r : x_1 \in \{0,1\}^*\}. \tag{4}$$
$$L_1 = \{x_1x_1 : x_1 \in \{0,1\}^*\}. \tag{5}$$
$$L_{2t} = \{x_1x_2 \dots x_{2t}x_1x_{2t}^r x_2x_{2t-1}^r \dots x_t x_{t+1}^r : x_i \in \{0,1\}^*\}. \tag{6}$$
$$L_{2t+1} = \{x_1x_2 \dots x_{2t+1}x_1x_{2t+1}^r x_2x_{2t}^r \dots x_{t-1}x_{2t}^r x_{t+1} : x_i \in \{0,1\}^*\}. \tag{7}$$

As examples we have

$$L_2 = \{x_1x_2x_1x_2^r : x_1, x_2 \in \{0,1\}^*\}, \tag{8}$$
$$L_3 = \{x_1x_2x_3x_1x_3^r x_2 : x_1, x_2, x_3 \in \{0,1\}^*\}. \tag{9}$$

Theorem 3.2 *For each $k \geq 0$, $L_k \in \mathcal{L}(k)$.*

Proof : Clearly L_0 is a context free language and hence is in $\mathcal{L}(0)$. By Proposition 3.2, we have $L_1 \in \mathcal{L}(1)$. Thus we consider the case $k \geq 2$.

Let $\mathcal{F}$ be the FPDA defined in the proof of Proposition 3.2. Note that we have not used the fact $\Delta(q_2) = \{q_2\}$ in the proof of Proposition 3.2. The condition $\Delta(q_2) = \{q_2\}$ allows the FPDA to reverse the stack while in state q_2. When we allow more than one stack reversals this is going to be important. Suppose x is accepted by $\mathcal{F}$ using exactly k stack reversals. Let $x = uv_1v_2 \dots v_k$, where u is the string that is read before the first stack reversal and for each $i \geq 1$, the string v_i is the string read between the ith and the $(i+1)$th stack reversals. Since u is the string read before the first stack reversal, the stack contains u before the first stack reversal. After the first stack reversal the FPDA goes to state q_2 and into the pop mode. Hence no additional symbol is pushed to the stack. In state q_2, the FPDA matches the input symbol with the stack top and pops the stack top if both are equal. Thus while v_i is being read, a portion of the stack is erased. We now determine the portion of the stack that is erased while v_i is being read. Let $u = u_1 \dots u_k$. Before the first stack reversal the stack is $u_1 \dots u_k$. After the first stack reversal, the stack is $u_k^r \dots u_1^r$. The string v_1 is matched to the stack and the string u_1^r is removed from the stack. This matching is successful iff $v_1 = (u_1^r)^r = u_1$. Thus the stack contains $u_k^r \dots u_2^r$ before the second stack reversal. After the second stack reversal the stack contains $u_2 \dots u_k$. The string v_2 is read and the string u_k is removed from the stack. Again this is possible if $v_2 = u_k^r$. Now it is not difficult to see that

$$\begin{aligned} v_i &= u_{\frac{i+1}{2}} \text{ if } i \text{ is odd.} \\ &= u^r_{k+1-\frac{i}{2}} \text{ if } i \text{ is even.} \end{aligned}$$

Thus $x \in L_k$. Conversely, given any string $x \in L_k$, it is possible to get a sequence of transtions of $\mathcal{F}$ using exactly k stack reversals and accepting x. Thus $L_k = L(\mathcal{F}(k))$ and hence $L_k \in \mathcal{L}(k)$. ■

4 Open Problems

We list some problems about FPDA's. The first problem would be to show that the hierarchy defined in (3) is strict.

1. Show that $L_k \notin \mathcal{L}(k-1)$, which would show that the hierarchy defined in (3) is strict.

2. Is there any other way to show that the hierarchy defined in (3) is strict ?

3. One can define the deterministic version of flip stack pushdown automata. This would provide a deterministic hierarchy similar to that of (3). The base of the hierarchy would change to the deterministic context free languages whereas the end of the hierarchy would still remain the recursively enumerable languages. Is the deterministic hierarchy strict ?

4. Can this notion of automata be used to model interesting natural language characteristics. A related problem is that of designing parsers for the deterministic FPDA which can make exactly k flips.

5. How does the deterministic and nondeterministic hierarchy intersect each other ?

6. The above hierarchy is formed by restricting the number of flips to constant. One can define language classes by allowing polynomial many flips, exponentially many flips and get analogous language classes. Is the corresponding hierarchy still strict ? We conjecture that it is indeed strict.

7. What is the relationship between these language classes and the known language classes ?

5 Conclusion

We have introduced the idea of allowing a pushdown automata to flip its stack. By bounding the number of flips it is possible to get a hierarchy from the context free languages to the recursively enumerable languages. We have shown that each class in the hierarchy is nonempty and conjecture that the hierarchy is strict. We believe that the language L_k can be used to separate $\mathcal{L}(k)$ from $\mathcal{L}(k-1)$. However, at this point we are not sure how this can be proved. Some interesting problems related to this kind of automata has been mentioned. Apart form the points of technical interest, it seems that this kind of pushdown automata can perhaps be used to model natural language characteristics. As a concluding remark, it must be stated that this work raises more questions than it answers. We hope that these questions will stimulate the interest of the research community.

References

[1] J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979.

[2] S. Rao Kosaraju. Real-time simulation of concatenable double-ended queues by double-ended queues (preliminary version). *In Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 346-351, Atlanta, Georgia, 30 April-2 May 1979.

[3] S. Rao Kosaraju. An optimal RAM implementation of catenable min double-ended queues. *In Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 195-203, Arlington, Virginia, 23-25 January 1994.