Introducing the pushdown operator: a proposal to extend regular expressions to context-free languages

Alexandre Tadeu Rossini da Silva[1], **Guilherme da Cruz Oliveira**[1], **Tanilson Dias dos Santos**[1]

[1]Department of Computer Science
Federal University of Tocantins (UFT) – Palmas – TO – Brazil

{arossini, guilherme.cruz1, tanilson.dias}@uft.edu.br

Abstract. *This research proposes a new operator, called the pushdown operator, to be used in combination with the classical operators of regular expressions and aims to increase the denotational power of regular expressions in order to express the set of context-free languages. A literature review was conducted covering the main concepts related to formal languages and denotational formalisms, with an emphasis on context-free languages, which serve as a theoretical basis to support this research. Finally, the formal definition of the pushdown operator is presented, along with comparisons to the other two denotational formalisms of context-free languages found in the literature.*

1. Introduction

Regular expressions, introduced by [Kleene 1956], are a widely adopted formalism for representing patterns in strings. They serve as the foundation for various applications, including lexical analysis, search algorithms, and automated text processing. Furthermore, by denoting language patterns, expressions are a formalism that enables efficient and flexible search, substitution, extraction, and validation [Thompson 1968, Gibney and Thankachan 2021, Takeshige et al. 2022, Sun et al. 2023]. Despite their utility, regular expressions are inherently constrained by their inability to capture recursive and hierarchical structures, which are essential in many computational contexts, such as programming language parsing and natural language processing (NLP). Context-Free Languages (CFLs), which lie above regular languages in the Chomsky hierarchy [Chomsky 1956], allow for the representation of these structures but traditionally require context-free grammars (CFGs) for their definition.

The development of the pushdown operator positions this work at the boundary of known language representation methods, extending classical formalism while preserving computational feasibility. Traditional approaches to formal languages, as discussed in [Hopcroft and Ullman 1979] and [Menezes 2000], establish the theoretical foundations of automata and formal grammars, defining the expressive limits of regular and CFLs. Then, [Sipser 2006] further refines these discussions by examining computability constraints within language classification. By introducing a structured extension to regular expressions, this research provides a novel approach to denoting CFLs and expands the expressive capacity of denotational methods.

Previous efforts to extend regular expressions for CFLs include μ-Regular expressions [Leiß 1991] and regular expressions with subpattern recursion [HruŠa 2021].

2. The Pushdown Operator

In this section, we introduce the new operator that, in combination with classical regular expression operators, can precisely denote the class of CFLs. This new operator is first introduced in Definition 1, in its base case, and has been named the pushdown operator. It is worth noting that the proposed operator, in its base case, is ternary, meaning it operates on three elements or operands to produce a result. These operands can be symbols from an alphabet or, in general, regular expressions with pushdown operators.

Definition 1. *The pushdown operator, in its base case, is represented as follows:*

$$a{:}b\#c \tag{1}$$

where a, b, c are regular expressions or expressions extended by the pushdown operator.

Definition 2. *The language denoted by the pushdown operator, in its base case, is defined as follows:*

$$L(a{:}b\#c) = \bigcup_{i=0}^{\infty} L(b^i a c^i) \tag{2}$$

where a, b, c are regular expressions or expressions extended by the pushdown operator.

Definition 3. *Let G be a CFG. The production rules of G, represent the same language as are given in Definition 2:*

$$S \to BSC \,|\, A \tag{3}$$

where A, B, and C are nonterminals that derive the expressions a, b, and c, respectively.

As has been said, the pushdown operator, in its base case, is ternary, which means the operation requires three operands a, b, and c. Therefore, in the following are listed the use cases of operands $a \neq \{\varepsilon\}$, $b \neq \{\varepsilon\}$, and $c \neq \{\varepsilon\}$ combined with the empty word ε.

- $\varepsilon{:}\varepsilon\#\varepsilon$
- $a{:}\varepsilon\#\varepsilon$
- $\varepsilon{:}b\#\varepsilon$
- $\varepsilon{:}\varepsilon\#c$
- $a{:}b\#\varepsilon$
- $a{:}\varepsilon\#c$
- $\varepsilon{:}b\#c$
- $a{:}b\#c$

Definition 4. *The pushdown operator has higher precedence.*

When using the pushdown operator with concatenation, union, and Kleene closure, it has a higher priority and, therefore, should be resolved before them. Some examples of this are presented in practice.

- $c{:}a\#a + b = (c{:}a\#a) + b$
- $\varepsilon{:}c\#b^* = (\varepsilon{:}c\#b)^*$
- $cb{:}a\#d = c(b{:}a\#d)$
- $a{:}b\#c{:}d\#e = (a{:}b\#c){:}d\#e$

Subsequently, we present the first modification to the base form of the operator. For example, the base form of the pushdown operator cannot denote the languages generated by this grammar $G = (V, T, P, S)$, where $V = \{S\}$, $T = \{a, b, c\}$, $P = \{S \to aSbSc \,|\, \varepsilon\}$, and $S = S$, as a single pushdown call is insufficient for multiple occurrences of S within a production. Since the number of occurrences can be any natural number n, a mechanism to represent multiple calls is essential. This necessity leads to the first proposed modification, formally presented in Definition 5.

Definition 5. *The pushdown operator with multiple calls is represented as follows:*

$$a{:}b_1 \# b_2 \# b_3 \# ... \# b_n \tag{4}$$

where a and b_i are regular expressions or expressions extended by the pushdown operator, n is a finite integer number ≥ 2 ($n = 2$ is the base case of the pushdown operator), and $2 \leq i \leq n$.

Definition 6. *Consider $\{a, b_1, b_2, \ldots, b_n\}$ as regular expressions or expressions extended by the pushdown operator, and let G be a CFG with nonterminals $\{A, B_1, B_2, \ldots, B_n\}$ such that $L(A) = L(a)$ and $L(B_i) = L(b_i)$ for $1 \leq i \leq n$. Then,*

$$L(a{:}b_1 \# b_2 \# b_3 \# b_4 \# ... \# b_n) = L(S \rightarrow A \,|\, B_1 S B_2 S B_3 S B_4 S ... S B_n\,) \tag{5}$$

Next, we present an extension to the pushdown operator. Throughout the development of its formalization, we have established its capability to represent a broad class of CFLs. However, we identified a fundamental limitation: certain languages do not admit a direct representation under the current definition. In particular, the operator, as currently defined, does not inherently support what we refer to as 'alternate calls", situations that occur in production rules of a CFG G such as the one below, where a variable calls itself recursively through two or more distinct derivation paths:

$$S \rightarrow bSb \;\mid\; cScSc \;\mid\; a \tag{6}$$

From this question, we propose a solution: extend the pushdown operator to encompass this infinite number of alternate calls into a single expression. The extension of the pushdown operator is in Definition 7.

Definition 7. *The pushdown operator was extended to cover cases in which there are altered recursive calls to this operator. The extension of the pushdown operator is represented as follows:*

$$L(a{:}(b_{1,1} \# b_{1,2} \# ... \# b_{1,n_1} \;;\; b_{2,1} \# b_{2,2} \# ... \# b_{2,n_2} \;;\; ... \;;\; b_{m,1} \# b_{m,2} \# ... \# b_{m,n_m})) \tag{7}$$

where the symbol ";" extends the pushdown operator by separating the groups ($b_{i,j} \# ... \# b_{i,n_i}$) and $b_{i,j}$ are regular expressions or expressions extended by the pushdown operator, which includes cases with multiple calls and the extension itself; n_i is an integer number ≥ 2 and $2 \leq i \leq n$; m is an integer ≥ 1 that determines the number of altered recursive calls of the operator such that $1 \leq j \leq m$ ($m = 1$ is equivalent to the pushdown operator with multiple calls).

Definition 8. *Consider $\{a, b_1, c_1, b_2, c_2, \ldots, b_n, c_n\}$ as regular expressions or expressions extended by the pushdown operator, and let G be a CFG with nonterminals $\{A, B_1, C_1, B_2, C_2, \ldots, B_n, C_n\}$ such that $L(A) = L(a)$, $L(B_i) = L(b_i)$, and $L(C_i) = L(c_i)$ for $1 \leq i \leq n$. Then,*

$$L(a{:}(b_1 \# c_1; b_2 \# c_2; ...; b_n \# c_n)) = L(S \rightarrow A \,|\, B_1 S C_1 \,|\, B_2 S C_2 \,|\, \ldots \,|\, B_n S C_n) \tag{8}$$

2.1. Languages Denoted by the Pushdown Operator

The regular expression with pushdown operator has the characteristic that, unlike the other two notations, it does not require variable declarations or parenthesis numbering, making it cleaner. This feature makes it simpler to design and more readable. Additionally, it still makes sense to use all the classical operators of regular expressions in the context-free expression, something that may contribute to its wider acceptance by the community. Next, in Table 1[1], are presented 14 CFLs denoted in the μ-regular expression, regular expression with subpattern recursion, and context-free expression.

Table 1. Examples of CFLs denoted in different representations.

Context-free language	μ-regular expression	Regular expression with subpattern recursion	Context-free expression (regular expression with pushdown operator)
$L_1 = \{a^n b^n \mid n \geq 0\}$	$\mu x.axb + 1$	$((a(?1)b) + \varepsilon)$	$\varepsilon{:}a\#b$
$L_2 = \{a^n b^n \mid n \geq 1\}$	$\mu x.axb$	$((a(?1)b) + ab)$	$(ab){:}a\#b$
$L_3 = \{a^{2n} b^n \mid n \geq 0\}$	$\mu x.aaxb + 1$	$((aa(?1)b) + \varepsilon)$	$\varepsilon{:}(aa)\#b$
$L_4 = \{a^{2n} b^n \mid n \geq 1\}$	$\mu x.aaxb$	$((aa(?1)b) + aab)$	$(aab){:}aa\#b$
$L_5 = \{a^n b^m \mid n \leq m\}$	$\mu x.(ax(\mu y.(by + 1)b) + 1)$	$(a(?1)(b(?2) + \varepsilon)b + \varepsilon)$	$(b^*){:}a\#b$
$L_6 = \{a^n b^m c^{n+1} \mid \{n, m\} \geq 0\}$	$\mu x.(ax(\mu y.(by+1)c)c+1)$	$(a(?1)(b(?2) + \varepsilon)c + \varepsilon)c$	$((b^*){:}a\#c)c$
$L_7 = \{a^n b^n c^m d^m \mid \{n, m\} \geq 0\}$	$\mu x.(axb + 1)\mu y.(cyd + 1)$	$((a(?1)b) + \varepsilon)((c(?3)d) + \varepsilon)$	$(\varepsilon{:}a\#b)(\varepsilon{:}c\#d)$
$L_8 = \{a^n b^m c^m d^n \mid \{n, m\} \geq 0\}$	$\mu x.(ax(\mu y.(byc+1)d+1))$	$((a(?1)(b(?3)c)d + \varepsilon) + \varepsilon)$	$(\varepsilon{:}b\#c){:}a\#d$
$L_9 = \{a^n b^m \mid n \geq 1,\ m \geq n\}$	$\mu x.(ax(\mu y.(by)b))$	$(a(?1)(b(?2))b + ab)$	$(abb^*){:}a\#b$
$L_{10} = \{a^{2n} b^{3n} \mid n \geq 0\}$	$\mu x.(aaxbbb + 1)$	$(aa(?1)bbb + \varepsilon)$	$\varepsilon{:}(aa)\#(bbb)$
$L_{11} = \{w \,\epsilon\, \{a, b\} \mid w$ has an odd length and contains a in the middle$\}$	$\mu x.(x(a+b+1))a\mu x.((a+b+1)x)$	$((a + b)(?1)(a + b) + a)$	$a{:}(a + b)\#(a + b)$
$L_{12} = \{w \,\epsilon\, \{a, b\} \mid w$ is an even-length palindrome$\}$	$\mu x.(axa + bxb + 1)$	$(a(?1)a + b(?1)b + \varepsilon)$	$\varepsilon{:}(a\#a; b\#b)$
$L_{13} = \{w \,\epsilon\, \{a, b\} \mid w$ is an odd-length palindrome$\}$	$\mu x.(axa + bxb + a + b)$	$(a(?1)a + b(?1)b + a + b)$	$(a + b){:}(a\#a; b\#b)$
$L_{14} = \{w \in \Sigma^* —$ all prefixes of w contain no more $]'s$ than $['s$ and the number of $['s$ in w equals the of $]'s\}$	$\mu x.([x]x + 1)$	$([(?1)](?1) + \varepsilon)$	$\varepsilon{:}[\#]\#\varepsilon$

3. Concluding Remarks and Future Works

In conclusion, this work lays the groundwork for future exploration and investigation by presenting a series of conjectures that are ripe for investigation. These conjectures not only open new branch of research, but also challenge the academic community to seek evidence and proofs that can either validate or refute them.

Conjecture 3.1. *Every CFL can be represented by pushdown operator expressions.*

Conjecture 3.2. *Every pushdown operator expression denotes a CFL.*

Conjecture 3.3. *The set of CFL is closed under the pushdown operator.*

The proofs of the conjectures aim to demonstrate that the pushdown operator precisely denotes the set of context-free languages.

[1]We sincerely thank Professor Ing. Ondrej Guth, supervisor of [HruŠa 2021], for reviewing the languages denoted by regular expressions with subpattern recursion in Table 1.

References

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

Gibney, D. and Thankachan, S. V. (2021). Text indexing for regular expression matching. *Algorithms*, 14(5):133.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

HruŠa, V. (2021). Regular expressions with subpattern recursion. Master's Thesis. Department of Theoretical Computer Science, Faculty of Information Technology CTU in Prague, Prague.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. *Automata studies*, 34:3–41.

Leiß, H. (1991). Towards kleene algebra with recursion. *Lecture Notes in Computer Science*, 626:242–256.

Menezes, P. B. (2000). *Linguagens formais e autômatos*. Sagra Luzzatto.

Sipser, M. F. (2006). *Introduction to the theory of computation (2nd ed.)*. Boston: Thomson Course Technology.

Sun, X., Mo, D., Wu, D., Ye, C., Yu, Q., Cui, J., and Zhong, H. (2023). Efficient regular expression matching over hybrid dictionary-based compressed data. *Journal of Network and Computer Applications*, 215:103635.

Takeshige, H., Matsumoto, S., and Kusumoto, S. (2022). Resem: Searching regular expression patterns with semantics and input/output examples. In *International Conference on Product-Focused Software Process Improvement*, pages 511–517. Springer.

Thompson, K. (1968). Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422.