Assignment 2

Dạ em chào thầy em tên Bùi Duy Nguyễn, em xin phép được sửa lại những lỗi thầy đã chỉ ra trong quá trình trình bày còn sai sót, thật sự đối với em môn này rất hay thiết kế trình biên dịch, và rất rất quan trọng đặc biệt với em. Em cảm ơn thầy nhiều.

Cho biết họ tên và mã số của bạn.

Sinh việc làm bài trang kế bên.

Bài toán: Viết chương trình tính toán một biểu thức số học.

Ví dụ: (2 + 5) * 10 -> 70

Trình bày giải thuật thực hiện. Đánh giá độ phức tạp của giải thuật.

HẾT

BÀI LÀM

MSSV 1:B2110133	Họ tên 1: Bùi Duy Nguyễn
MSSV 2:B2012183	Họ tên 2: Văn Thị Thanh Diễm

Theorem: If G is a Context Free Grammar for a language L, then there exists a PDA for L as well.

Based on the theorem given, we try to construct a Push Down Automata (PDA) which validates Context Free Grammar (CFG), so that we can validate and calculate the valid arithmetic expression.

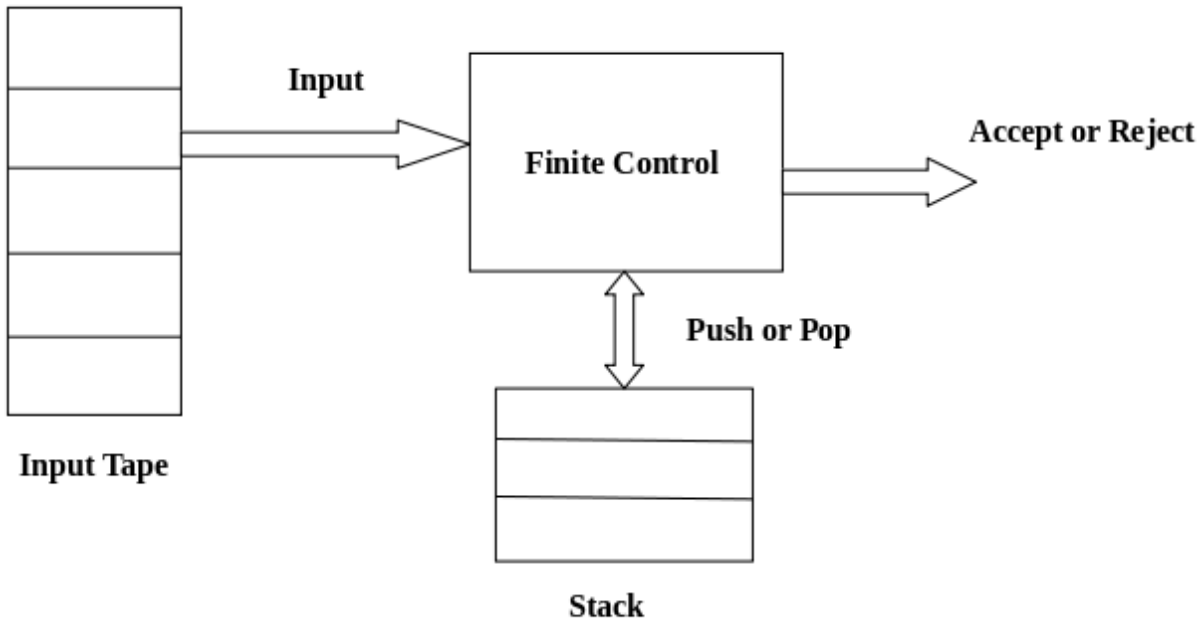


Fig: Pushdown Automata

PDA Example

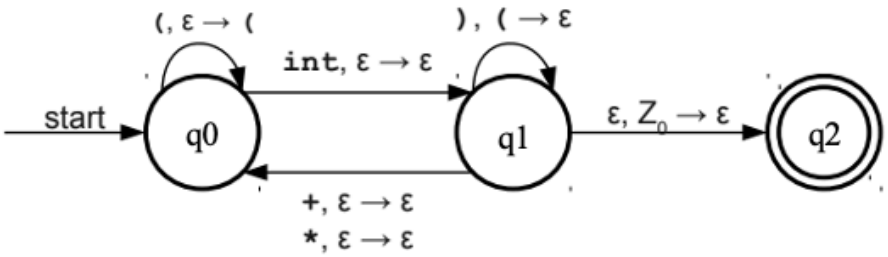
Reference: https://web.njit.edu/~marvin/cs341/projects/prog-ae-hints.html

Based on the given expression, I try to construct a language with

∑ = { **int, +, *, (,)** } and a given language

L = { w ∈ Σ* | w is a valid arithmetic expression } for example (int+int)*int

PDA Transition Diagram



So that, we can generate code using this PDA 😀 C# is my favorite language. For these reasons, I prefer using C# to demonstrate this algorithm. Please forgive me if I make mistakes when demonstrating this problem.

After I calculated by hand, this string has been accepted by not only the PDA but also the CFG.

PDA = (**Q:** the finite set of state, ∑**:** the input set **, Γ:** a stack symbol which can be pushed and popped from the stack, **q0:** the initial state, **Z:** a start symbol which is in Γ , **F:** a set of final states, **δ:** mapping function which is used for moving from current state to next state)

Instantaneous Description (ID) = a triple (**q** describes the current state, **w** describes the remaining input, **α** describes the stack contents, top at the left)

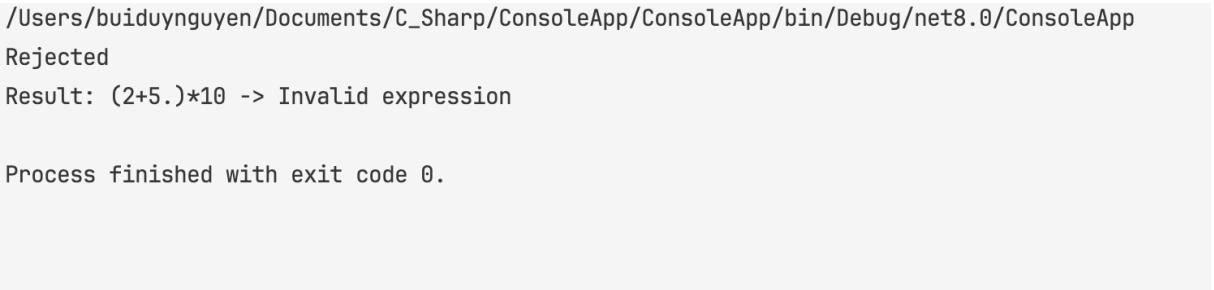
Source code can be found here: https://github.com/duynguyenbui/PDA

```
// See https://aka.ms/new-console-template for more information
/*
 * Number can be replaced by int for general, but for programing convenience I prefer using real number for testing
 * Author: Duy Nguyen BUI responsible for coding, developing and testing this algorithm
 */

using System.Globalization;
using System.Text.RegularExpressions;

var numbers = Enumerable.Range(1, 10).Select(i => i.ToString()).ToList();
List<string> alphabet = [..numbers, "+", "*", "(", ")", "N"];
var pda = new PDA(alphabet);
const string input = "(2+5)*10";
Console.WriteLine(pda.Run(input) ? "Accepted" : "Rejected");
Console.WriteLine(
    $"Result: {input} → {(input.EvaluateExpression(input) == 0 ? " Invalid expression" : input.EvaluateExpression(input).ToString(CultureInfo.CurrentCulture))}" );

/* Class Push Down Automata for the given CFG */
public class PDA
{
    private string state;
    private Stack<string> stack;
    private List<string> alphabet;

    public PDA(List<string> alphabet)
    {
        state = "q0"; stack = new Stack<string>();
        stack.Push("Z"); this.alphabet = alphabet;
    }

    private void Transition(string input)
    {
        if (!alphabet.Contains(input)) throw new Exception("Missing alphabet for this input");
        switch (state)
        {
            case "q0" when input == "(":
                stack.Push("(");
                break;
            case "q0" when int.TryParse(input, out _):
                state = "q1";
                break;
            case "q1" when input == ")":
                if (stack.Pop() == "(") state = "q1";
                break;
            case "q1" when input == "N":
                state = "q2";
                break;
            case "q1" when input is "+" or "*":
                state = "q0";
                break;
        }
    }

    public bool Run(string input)
    {
        try
        {
            const string pattern = @"(\d+|\D)";
            var matches = Regex.Matches(input + "N", pattern);
            foreach (var match in matches.ToList()) Transition(match.Value);
            return stack.Pop() == "Z" && state == "q2";
        }
        catch (Exception)
        {
            return false;
        }
    }
}
```

PDA Implementation

Evaluate after validate language expression

The given string accepted by the PDA



The given string rejected by the PDA

Đánh giá độ phức tạp của giải thuật:

Thuật toán chia làm 2 phần một phần dùng để xác định chuỗi đã cho có được chấp nhận bởi ngôn ngữ CFG đã cho không và một phần tính giá trị biểu thức dựa trên chuỗi đó.

Phần 1 thuật toán sử dụng cấu trúc dữ liệu Stack cũng như vòng lặp foreach nên có thể nói độ phức tạp của thời gian là **O(n)**, với n là số lượng ký tự trong đầu vào. Tuy nhiên, khi xác định thuật toán và sơ đồ PDA, độ phức tạp thời gian có thể phụ thuộc vào cách thức ánh xạ các quy tắc của ngôn ngữ CFG thành trạng thái và chuyển tiếp trên Stack. Trong trường hợp này, việc xác định chính xác độ phức tạp thời gian sẽ phụ thuộc vào chi tiết cụ thể của quy tắc sinh và cách mà chúng được thực hiện trong thuật toán. Sơ đồ PDA xác định cách mà các trạng thái của PDA chuyển đổi dựa trên các ký tự đầu vào và trạng thái hiện tại của Stack. Một sơ đồ PDA cụ thể có thể có một cấu trúc và một số quy tắc chuyển đổi khác nhau, phụ thuộc vào ngôn ngữ mà nó đang cố gắng chấp nhận. Do đó, độ phức tạp của thuật toán sẽ phụ thuộc vào sơ đồ PDA cụ thể được sử dụng.

Phần hai thuật toán sử dụng hai đối tượng Stack để giải quyết bài toán tính toán giá trị của một biểu thức toán học trong chuỗi. Stack được sử dụng để lưu trữ các số và các toán tử trong biểu thức, giúp đảm bảo tính đúng đắn của phép tính. Trong quá trình duyệt qua từng ký tự trong biểu thức, các số được đẩy vào Stack numbers, trong khi các toán tử được đẩy vào Stack operators. Các phép toán thực hiện trên Stack như Push, Pop, và Peek giúp kiểm soát thứ tự và ưu tiên của các phép toán. Mỗi lần gặp một toán tử hoặc dấu ngoặc đóng, thuật toán thực hiện các phép toán để tính toán giá trị của các số và toán tử tương ứng trong Stack. Khi gặp dấu ngoặc đóng, các phép toán được thực hiện cho đến khi gặp dấu ngoặc mở tương ứng. Độ phức tạp của thuật toán thứ 2 là **O(n)**.