Tecnológico de Monterrey

Math expressions

Santiago Alonzo Aguilar A01639373

Hannia Escamilla Pérez A01639113

Implementación de métodos computacionales (Gpo 603)

Obed Nehemías Muñoz Reynoso

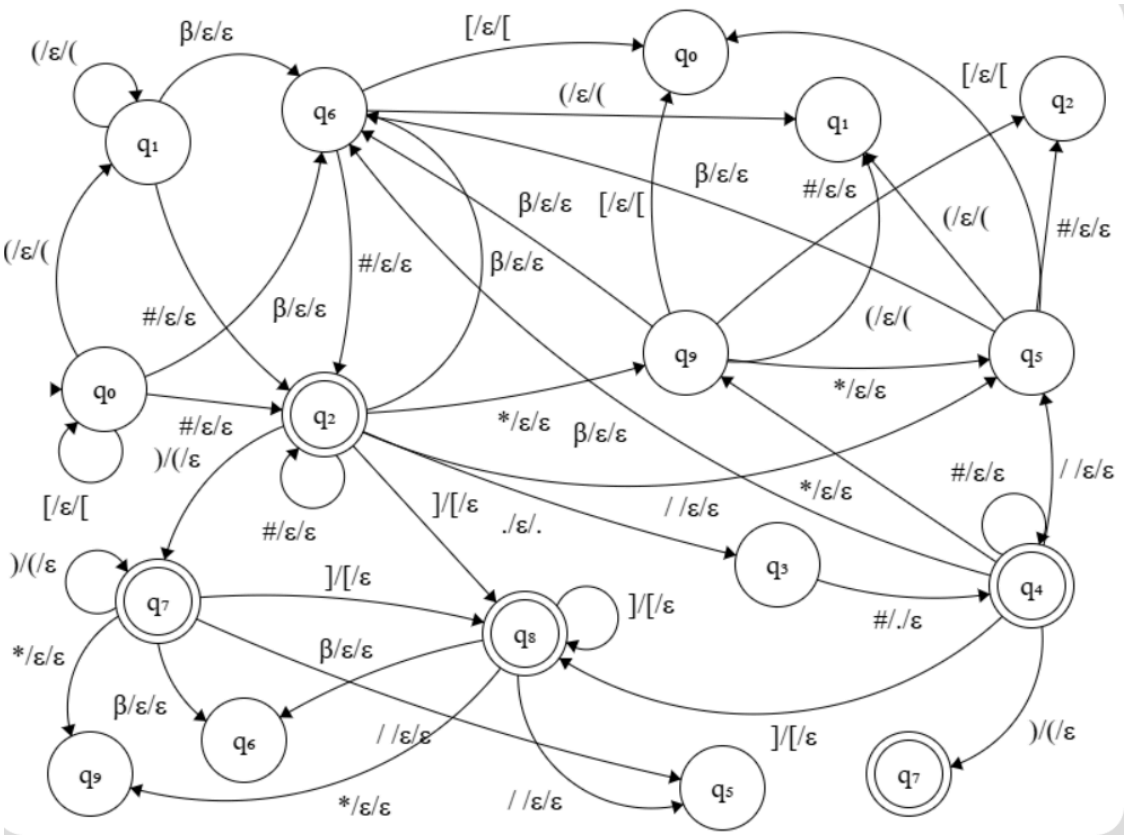
25 de Mayo del 2025

Tecnológico de Monterrey

DESCRIPTION OF THE PROJECT

La meta del proyecto era crear un analizador de expresiones matemáticas que pudiera revisar si ciertas expresiones matemáticas simples eran válidas o no, basándonos en un lenguaje específico.

Diagrama de PDA



En el comienzo se iba a utilizar un PDA y 2 DFA. Los DFAs iban a ser utilizados para revisar números y operadores mientras que el PDA iba ser usado para revisar los paréntesis y corchetes. Se iban a tokenizar los strings o cadenas para pasarlos de los DFAs y del PDA para validar, pero esto hacía más complicado el proceso. Al final para simplificar todo se juntó en un PDA con 9 estados.

Nuestra gramática incluye:

- ★ Paréntesis () y corchetes []
- ★ Operadores:
  - Multiplicación *
  - Potencia **
  - División /
  - Suma +
  - Resta -
- ★ Números reales (0, 1, -2, +7, -8.5, +7.8, 4.1546, …)

Explicacion del codigo

Estructura de datos

Para el manejo del PDA y sus resultados se usó la siguiente estructura. En el PDA transiciones es un arreglo de transiciones ya que cada transición es muy compleja para guardarse de alguna otra manera.

Go

```go
type PDA struct {
    InitialState  string       `yaml:"S"`
    States        []string     `yaml:"K"`
    Alphabet      []string     `yaml:"E"`
    StackAlphabet []string     `yaml:"R"`
    Transitions   []Transition `yaml:"T"`
```

```
    FinalStates   []string     `yaml:"F"`
}
type Transition struct {
    State     string `yaml:"state"`
    Input     string `yaml:"input"`
    StackPop  string `yaml:"stack_pop"`
    ToState   string `yaml:"to_state"`
    StackPush string `yaml:"stack_push"`
}
type Result struct {
    Num        int    `yaml:"num"`
    Expression string `yaml:"expression"`
    Valid      bool   `yaml:"valid"`
}
type Stack struct {
     items []string
}
```

Procesamiento de datos

Dentro de la función para analizar cada cadena primero limpiamos los espacios para evitar leerlos. Seguido de esto revisamos que cada carácter esté dentro de nuestro alfabeto, en caso de no ser así rompe la ejecución del for loop y lo marca “valid” como falso.

Go

```go
func analizeString(pda PDA, expression string, num_expression int)
(result Result) {
    stack := stack.New()
    valid := false
    cleanStr := strings.ReplaceAll(expression, " ", "")
    // validate alphabet. if all characters are in alphabet the for
loop will set valid to true
    for _, char := range cleanStr {
        valid = false
        for _, charA := range pda.Alphabet {
            if string(char) == charA {
                valid = true
                break
            }
        }
        if !valid {
            break
        }
    }
```

Si “valid” es verdadera se ejecuta la validación de transiciones en “executeTransition”. En el caso de que no encuentre una transición válida “valid” se volverá falsa y dejará de revisar transiciones. Una vez validadas las transiciones revisa si el estado en el que se quedó la última transición sea una final solamente si “valid” es verdadera. Finalmente se guardan los resultados.

Go

```go
    //validate with transitions, not even bother if alphabet is
invalid
    currentState := pda.InitialState
    if valid {
        //check every char in string
```

```
        for _, char := range cleanStr {
            currentState, valid = executeTransition(pda,
currentState, string(char), stack)
            if !valid {
                break
            }
        }
    }
    //check final state
    if valid && isFinalState(currentState, pda.FinalStates) &&
stack.IsEmpty() {
        valid = true
    } else {
        valid = false
    }
    result.Num = num_expression
    result.Expression = expression
    result.Valid = valid
    return result
}
```

Lógica de transiciones

Por el carácter a evaluar revisamos cada transición posible hasta que encontremos una con ese carácter. Es muy importante que en el diseño del PDA no se tengan más de 1 transición con el mismo carácter a leer en el mismo estado debido a que solo leerá la primera y se perderán las demás.

Una vez encontrada una transición, si se requiere, quitamos del stack un valor y evaluamos que sea el correcto. En caso de un valor incorrecto detenemos la ejecución enviando

falso a “valid”. Con un valor correcto en caso de ser necesario ponemos un valor en la pila y terminamos la función enviando el nuevo estado y verdadero a “valid”

Go

```go
func executeTransition(pda PDA, currentState string, inputChar
string, stack *stack.Stack) (string, bool) {
    for _, trans := range pda.Transitions {
        if currentState == trans.State && inputChar == trans.Input {
            // Handle stack pop
            if trans.StackPop != "ε" {
                if stack.IsEmpty() {
                    return currentState, false
                }
                popped, _ := stack.Pop()
                if popped != trans.StackPop {
                    return currentState, false
                }
            }
            if trans.StackPush != "ε" {
                stack.Push(trans.StackPush)
            }
            // Transition executed crrectly
            return trans.ToState, true
        }
    }
    // No valid transition found
    return currentState, false
}
```

Transiciones de números

Al leer el archivo yaml, cuando leemos una transición de número en vez de escribir 10 veces la misma transición por cada número el programa reconoce “num”. Al crear las

transiciones si se encuentra un “num” se reemplaza automáticamente con 10 transiciones iguales excepto el input que son números del 0 al 9.

```Go
switch t.Input {
        case "num":
            for i := 0; i <= 9; i++ {
                expandedTransitions = append(expandedTransitions, Transition{
                    State:     t.State,
                    Input:     fmt.Sprintf("%d", i),
                    StackPop:  t.StackPop,
                    StackPush: t.StackPush,
                    ToState:   t.ToState,
                })
            }
        default:
            expandedTransitions = append(expandedTransitions, t)
        }
```

Ejemplo de ejecución

Al ejecutar el siguiente comando se cargan 2 archivos, un pda en formato yaml y un archivo de texto. El formato del archivo de texto es solamente una cadena de texto a evaluar por línea.

```Unset
go run pda.go -pda [yourPdaFile.yaml] -input [yourInputFile.txt]
```

El formato del yaml es el siguiente:

```Unset
S: q0 #initial state
```

```
K: [q0, q1, q2, q3, q4, q5, q6, q7, q8, q9] #set of states
E: ["[", "]", "(", ")", "*", "/"] #alphabet
R: ["[", "(", "."] #alphabet stack
F: [q2, q4, q7, q8] #final states
T: #transitions
  - state: q0
    input: "["
    stack_pop: ε
    stack_push: "["
    to_state: q0
```

El resultado saldrá en un archivo de texto llamado results.txt:

```
Unset
Expression 0: 5*7                  -  Valid
Expression 1: 5 * 5                -  Valid
Expression 2: [(5 + 5)/(6-4)]      -  Valid
Expression 3: (4***(3))            -  Invalid
```

Conclusions (one per team member)

Santiago Alonzo Aguilar

Me gustó mucho la estructura y funcionalidad de los PDA, siento que son muy intuitivos y fáciles de diseñar. Al principio de este proyecto intentamos hacer varios lenguajes para evaluar cosas distintas a la vez lo que haría el código más rápido sin embargo solo funcionaria en el contexto de expresiones matemáticas. Pero, terminamos decidiendo realizar una solución más universal para soportar varios lenguajes distintos. Esto nos ayudó mucho en cuanto a las pruebas del código porque podríamos cargar varios PDA muy simples y verificar que funcionan, lo que nos dio la certeza que el código funcionaba. Ahora lo único que nos preocupaba era que la lógica del PDA fuera correcta y no hubieran errores al escribirlos.

Tecnológico de Monterrey

Hannia Escamilla Pérez

Este proyecto se me hizo muy interesante porque fue la primera vez que vi temas de autómatas, me tomó mucho en realidad entender ciertos aspectos de los diferentes tipos de autómatas. Las limitaciones que tienen los DFAs en realidad hizo que entendiera más los autómatas de pila. Intentar simplificar más el diagrama del PDA y revisar las transiciones fue tedioso pero en un punto es fácil de seguir porque es como funciona un stack.