Stackmaschine eREGS

Jeff Wagner

May 13, 2017

Abstract

Erstellen einer Sprache für einen Stackautomaten[1] und eines in C geschriebenen Interpreters

1 Vorwort

Es wird eine Sprache, *"eREGS"* genannt, entworfen.
Des Weiteren wird mittels der Programmiersprache C und den Scanner- und Parsergeneratoren *"flex"*[2] respektive *"yacc/bison"*[3] ein Programm entworfen, welches eine Eingabe-Code-Datei interpretiert und ausführt.
Es wird **kein** Ausgabemodul geschrieben, welches z.B. den Code für eine spezifische Prozessorarchitektur übersetzt, auf derer das Kompilat einzig auszuführen wäre.

Bevor es in Abschnitt 3 ab Seite 3 mit einer detaillierten Erläuterung der Sprache weitergeht, wird im Folgenden noch das Code-Layout 2.1 und die Inbetriebnahme des Projektes 2.2 vorgestellt.

[1] https://de.wikipedia.org/wiki/Kellerautomat
[2] https://de.wikipedia.org/wiki/Lex_(Informatik)
[3] https://de.wikipedia.org/wiki/Yacc

2 Einleitung

Das Programm ist nach folgender Konvention nur von der Kommandozeile auszuführen:

Aufruf (Linux/MAC): ./eregs Code-Eingabedatei

Aufruf (Windows): eregs.exe Code-Eingabedatei

2.1 Code-Layout

Das Projekt hat folgenden Aufbau:

bin/	Ordner mit dem von der MAKEFILE erstelltem Programm
doc/	LaTeX Quellcode und PDF mit dieser Dokumentation
examples/	Beispielprogramme in der "eRegs" Sprache
main.l	Programm-Einstiegspunkt und Scanner-Definition
makefile	Automatische Programmerzeugung mit "make" tool
parser.y	Parser und Interpreter-Implementierung

2.2 Kompilierung & Programmablauf

Zum Kompilieren genügt es, mit der Kommandozeile in das Verzeichnis zu gehen, welches die "makefile" beinhaltet (also das Stamm-Projektverzeichnis) und ***make*** einzugeben. Anhand dessen wird bei erfolgreicher Kompilierung das Programm im ***bin***-Ordner erzeugt.
Um ein in "eRegs" geschriebenes Programm zu starten, muss der Interpreter mit der Code-Datei als Argument von der Kommandozeile gestartet werden.
Siehe dazu Abschnitt 2.

3 Die Sprache eREGS

Die Sprache "eRegs" unterstützt arithmetische Operationen sowie Verzweigungen und Schleifen.
In der ALPHA werden momentan bis zu 32 Register (quasi Variablen) und 999 Sprungmarken (für Verzweigungen und Schleifen) unterstützt.
Der Stack kann ebenso simultan maximal 999 Werte zwischenspeichern.
Des Weiteren werden innerhalb des Quelltextes Kommentare
im C Stil (`//` für Einzeilige Kommentare und `/* ... */` für Mehrzeilige Kommentare) unterstützt.

Der genaue Befehlssatz umfasst:

push	Legt Element auf die Spitze des Stacks
pop	Holt oberstes Element vom Stack
add	Holt die 2 obersten Elemente vom Stack, addiert diese und legt das Ergebnis auf den Stack
sub	Wie "add", nur für Subtraktion
mul	Wie "add", nur für Multiplikation
div	Wie "add", nur für Division (prüft Division mit Null)
cmpl	Prüft die 2 obersten Elemente (setzt 1, falls Arg1 < Arg2), 0 ansonten
cmpg	Prüft die 2 obersten Elemente (setzt 1, falls Arg1 > Arg2), 0 ansonten
cmple	Prüft die 2 obersten Elemente (setzt 1, falls Arg1 <= Arg2), 0 ansonten
cmpge	Prüft die 2 obersten Elemente (setzt 1, falls Arg1 >= Arg2), 0 ansonten
cmpeq	Prüft die 2 obersten Elemente (setzt 1, falls Arg1 == Arg2), 0 ansonten
cmpne	Prüft die 2 obersten Elemente (setzt 1, falls Arg1 ! = Arg2), 0 ansonten
print	Holt das oberste Element vom Stack und gibt es auf der Konsole aus
jmp	Unbedingter Sprung zu Sprungmarke **L***xxx*
jfalse	Bedingter Sprung zu Sprungmarke **L***xxx* (Prüft Stack)
L	Definition einer Sprungmarke, z.B. **L***1*:
r	Definition eines Registers, z.B. **r***1*
shl	Binärer *Linksshift* um n Stellen $(n * 2^n)$
shr	Binärer *Rechtsshift* um n Stellen $(n/2^n)$
rol	Binäre *Linksrotation*
ror	Binäre *Rechtsrotation*
not	Binäre *Negation*
and	Binäres *UND*
xor	Binäres *Exklusiv-ODER*
mod	Modulo Restoperator

Genug der Theorie, in Abschnitt 4 geht es nun an ein paar Beispiele.
Eine Definition der Grammatik der Sprache WIRD NOCH NACHGEREICHT...

4 Programmbeispiele

4.1 Do-While-Schleife

```
/*
  Folgendes Stackmaschinenprogramm durchläuft rückwarts eine
  Schleife von 5-0 und gibt den derzeitigen Iterationswert an.
*/
  push $5   // 5 auf den Stack legen
  pop r2    // r2 = 5, ToS = {}
L0:
  push r2   // ToS = 5
  print   // ToS = {}
  push $1   // ToS = 1
  push r2   // ToS = 5, 1
  sub   // ToS = 4
  pop r2    // r2 = 4, ToS = {}
  push r2   // ToS = 4
  push $0   // ToS = 0, 4
  cmpge
        jfalse L1
        jmp L0
L1:
```

../examples/example1.r

4.2 Additionsbeispiel

```
/*
  Folgendes Stackmaschinenprogramm addiert einfach nur ein wenig ;)
  Author: Jeff Wagner
  Date: 23.04.2017
  Aufruf: ./eregs example2.r
*/
push $0   // ToS = 0
push $22  // ToS = 22, 0
add   // ToS = 22
pop r1    // r1 = 22, ToS = {}
push r1   // ToS = 22
print   // ToS = {}
push r1   // ToS = 22
push $26  // ToS = 26
add   // ToS = 48
print   // ToS = {}
```

../examples/example2.r

4.3 Berechnung einer mathematischen Reihe

```
/*
  Dieses Beispiel berechnet folgende Reihe:
    {(n * n-1)+(n-1 * n-2)+(n-2 * n-3)... | n-... >= 0}
*/
  push $5    // ToS = 5
  pop r1     // r1 = 5, ToS = {}
L0:
  push r1    // ToS = 5
  push $1    // ToS = 1, 5
  push r1    // ToS = 5, 1, 5
  sub    // ToS = 4, 5
  mul    // ToS = 20
  push r2    // vorheriges Ergebnis auf den Stack legen (0 in der ersten
     Iteration)
  add    // addieren
  pop r2     // r2 = 20
  push $1    // ToS = 1
  push r1    // ToS = 5, 1
  sub        // ToS = 4
  pop r1     // r1 = 4, ToS = {}
  push r1    // ToS = 4
  push $1    // ToS = 1, 4
  cmpge      // ToS = 1
  jfalse L1    // ToS = {}, egal wie der Vergleich ausging
  jmp L0
L1:
  push r2
  print
```

../examples/example3.r

4.4 Berechnung der Fakultät $n!$

```
/*
  Dieses Beispiel berechnet die Fakultät einer als Startwert gegebenen Zahl.
    {(n * n-1 * n-2 * n-3...) | n >= 1}

  // Fakultät iterativ
  long long fakIter(int start) {
          long long res = 1;
          for(int i = start; i > 1; i--)
                  res *= i;
          return res;
  }

  r1 = Iterationszähler
  r2 = Ergebnis
*/
  push $5   // ToS = 5
  pop r1    // r1 = 5, ToS = {}
  push $1   // ToS = {}
  pop r2    // r2 = 1, ToS = {}
L0:
  push r1   // ToS = 5, 4, 3...
  push r2   // ToS = 1, 20...
  mul   // ToS = 5, 20...
  pop r2    // r2 = 5, 20..., ToS = {}
  push $1   // ToS = 1
  push r1   // ToS = 5, 1
  sub       // ToS = 4
  pop r1    // r1 = 4, ToS = {}
  push r1   // ToS = 4
  push $1   // ToS = 1, 4
  cmpg    // ToS = 1
  jfalse L1   // ToS = {}, egal wie der Vergleich ausging
  jmp L0
L1:
  push r2
  print
L2:
  /* DEBUG MARKER, PRINT FROM STACK AND JUMP TO THIS LOCATION TO
     GUARANTEE A SUCCESSFUL PROGRAMM RUN */
```

../examples/example5.r

4.5 Shift Beispiele

```
/*
  Dieses Beispiel demonstriert den Links-Shift,
  den Rechts-Shift, den NOT-Operator sowie den AND-Operator.
*/
  // Links-Shift
  push $1   // n
  push $6   // x
  shl   // entspricht x * 2^n
  print
  // "Fehlerfall"
  push $0
  push $6
  shl
  push $6
  cmpeq
  jfalse L1
  push $8
  push $2
  shl
  print

  // Rechts-Shift
  push $2   // n
  push $100 // x
  shr   // entspricht x / 2^n
  print

  // NOT
  push $0
  not
  print
  push $1
  not
  print
  push $5   // Nicht booleanscher Wert
  not
  print

  // AND
  push $1
  push $1
  and
  print
  push $123
  push $654
  and
  print
  jmp L2
L1:
  /* Sollte nie ausgeführt werden */
  push $-1
  print
L2:
  /* ENDE */
```

../examples/example6.r

4.6 Berechnung des Durchschnitts 2er Zahlen

```
/*
  This example calculates the average of two given integers.
  Date: 05/05/17
  hm89
*/
push $22
push $48
xor
pop r1
push $1
push r1
shr
pop r1
push $22
push $48
and
push r1
add
print
```

../examples/example9.r

Für weitere *eventuell vorhandene* Beispiele siehe das ***examples***-Verzeichnis.