Programmierpraktikum

Einfacher? Crosscompiler von Python nach C++

Felix Helsch
Julian Buchhorn

24. Januar 2015

Betreuer
Dr. Holger Arndt

Inhaltsverzeichnis

1 Einleitung

Im Rahmen eines Programmierpraktikums, an der Bergischen Universität Wuppertal, haben wir einen Compiler von Python zu C++ geschrieben. Dabei ging es in erster Linie darum, zu verstehen, wie ein Compiler funktioniert und nicht darum einen vollständigen Compiler zu schreiben.

Als vorläufiges Ziel haben wir uns gesetzt ein einfaches Testprogramm mit grundlegenden Syntaxelementen zu übersetzen (s. Lst. 1).

Listing 1: *Python Testprogramm für den Compiler*

```
1  from __future__ import division
2
3  """
4  Test program  for codegeneration
5  """
6
7  def f(x):
8      """calculate"""
9      return x**2 + x*4 - (x-3) /x -1.3e2+2\
10             +5 #comment
11
12
13
14 if __name__=='__main__':
15
16     L= [0]*10
17
18     for i in range(len(L)):
19         if i>3:
20             L[i]= f(i+1)
21         else:
22             L[i]+=1
23
24     for x in L:
25         print x
26
27     print 'program', 'end'
```

Das Ziel haben wir erreicht. Unser Compiler kann in seinem momentanen Zustand das Python Programm übersetzen und liefert das C++ Programm in Listing 2. Dabei mussten wir, wie erwartet, einige Annahmen über das Programm machen, um die Übersetzung zu vereinfachen.

Listing 2: *Vom Compiler erzeugtes C++ Programm*

```
1  #include <array>
2  #include <cmath>
3  #include <iostream>
4
5  using namespace std;
6
```

```
7  /*
8  Test program  for codegeneration
9  */
10
11 double f(double x) {
12     /*calculate*/
13     return pow(x, 2)+ x*4 - (x-3) /(double) x-1.3e2+2
14            +5; //comment
15 }
16
17
18
19 int main() {
20
21     array<double,10> L={};
22
23     for (int i= 0; i<L.size(); i++) {
24         if (i>3) {
25             L[i]= f(i+1);
26         }
27         else {
28             L[i]++; // com
29         }
30     }
31
32     for (auto x : L) {
33         cout<< x << endl;
34     }
35
36     cout<< "program" <<' '<< "end" << endl;
37
38     return 0;
39
40 }
```

In den folgenden Kapiteln werden wir erklären wie man den Compiler verwenden kann, welche Programme zur Entwicklung benötigt werden und welche Features der Compiler im Moment unterstützt. Zum Schluss werden wir noch kurz auf die Struktur des Codes eingehen, den wir geschrieben haben.

Für eine detailliertere Erklärung verweisen wir auf die Bachelorarbeit von Felix Helsch ..., in welcher die Erstellung des Compilers fortgeführt wird.

2 Installation

Um den Compiler ausführen zu können, muss Python in der Version 2.7 installiert sein. Außerdem werden die Python-Bibliotheken Numpy und Pandas, sowie deren Abhängigkeiten, benötigt. Am einfachsten ist es, hierfür einen Package-Manager wie zum Beispiel Anaconda zu installieren.

Wenn die enthaltene Python Grammatik geändert werden soll und diese somit neu compiliert werden muss, ist die Installation etwas aufwändiger. Im Folgenden werden die Installationsschritte für Windows und Linux gezeigt.

1. Installation des Java Development Kit (SDK/JDK) in der Version 1.6 oder höher
2. Download des „Complete ANTLR 4.4 Java binaries jar“-Paketes (oder neuer) von der Download-Seite des ANTLR-Projektes
3. Kopieren der heruntergeladenen Java-Bibliothek in den „ jre/lib/ext“-Ordner des JDK (zum Beispiel: „$C:\ProgramFiles(x86)\Java\jdk1.8.0\_25\jre\lib\ext$“ bzw. unter Linux: $/usr/lib64/jdk1.8.0\_25/jre/lib/ext/$)
4. Hinzufügen der jar-Datei zum Klassenpfad

3 Verwendung

Um den Compiler zu verwenden muss das Python-Script „py2cpp.py“ ausgeführt werden. Als Parameter muss das zu übersetzende Programm angegeben werden:

python py2cpp.py program.py

4 Features

5 Codestruktur

Abbildungsverzeichnis

Tabellenverzeichnis

Listings