Python2Java

Hubert Rędzia

Grzegorz Puczkowski

Spis Treści

1 Opis projektu

Jak nazwa wskazuje projekt ma na celu translację wejściowego pliku tekstowego z kodem języka Python do pliku wyjściowego w języku Java.

Sam projekt napisany jest w języku Java. Do stworzenia translatora skorzystaliśmy z narzędzia ANTLR4, które pomaga nam w wygenerowaniu Lexera oraz Parsera ze stworzonej wcześniej gramatyki języka Python.

1.1Zakres i cel projektu

- Stworzenie gramatyki języka Python, posiadającej zbiór symboli nieterminalych i terminalnych, reguły produkcji określających w jaki sposób program ma odczytywać plik wejściowy napisany w języku Python.
- Wygenerowanie za pomocą narzędzia ANTLR4 parsera i lexera dla języka
- Zbudowanie drzewa przejść, które reprezentuje jak nasza gramatyka odczytuje dane wejściowe – w tym przypadku skrypt napisany w języku Python.
- Translacja pliku wejściowego napisanego w języku Python, czyli określenie jak program ma się zachować podczas napotkania danej reguły w procesie przejścia drzewa parsowania

2 Narzędzia do generowania parserów

2.1 YACC

Program YACC jest generatorem analizatorów składniowych. Z zadanej specyfikacji program generuje kod źródłowy analizatora składniowego. Domyślnie tworzony jest kod źródłowy w języku C. Program został zaprojektowany w ten sposób, aby można go było wykorzystywać z generatorem analizatorów leksykalnych LEX.

Program generuje analizator redukujący działający w oparciu o tablicę LALR.

Nieterminal znajdujący się po lewej stronie pierwszej produkcji jest domyślnie wykorzystywany jako symbol startowy gramatyki.

2.2 Java CC

JavaCC to generator parserów i analizatorów leksykalnych. Podobnie jak ANTLR Java CC ze stworzonej gramatyki generuje analizator leksykalny, który dzieli program na tokeny. Te oczywiście określają słowa kluczowe, znaki, literały etc. W drugim kroku generuje parser, który podczas analizy składniowej wyodrębnia znaczenie , zapewniając logiczną reprezentację programu. Narzędzie to generuje interfejs, który trzeba później zaimplementować .

Zaletami JavyCC są :

- JJDoc, które automatycznie generuje dokumentację w HTMLu dla stworzonej gramatyki
- JJTree, które automatycznie generuje strukturę o charakterze drzewa podczas parsowania programu. Dostarcza ono również framework opierający się na wzorcu projektowym 'Visitor', które pozwala przemieszczać się po drzewie.
- Dobre raportowanie błędów
- Generuje parsery 'top-down' co pozwala na użycie bardziej ogólnych gramatyk

3 Wybrane narzędzie – ANTLR4

Wybrane przez nas narzędzie to ANTLR4. Jest to potężny mechanizm, który z opisu języka formalnego, czyli gramatyki potrafi wygenerować parser dla tego języka. Parser ten automatycznie buduje drzewa przejść. ANTLR również automatycznie generuje „tree walkers”, które można użyć do odwiedzenia węzłów, co znacznie pomaga w napisaniu kolejnych funkcjonalności dla odpowiednich reguł.

ANTLR używa technologii analizowania zwanej Adaptive LL (*) lub ALL (*) („all star”).Oznacza to, że wygenerowane parsery nie są analizowane statycznie, ale są generowane na bieżąco. Ponieważ parsery ALL (*) mają dostęp do rzeczywistych sekwencji wejściowych, zawsze mogą dowiedzieć się, jak rozpoznać sekwencje poprzez odpowiednie przeanalizowanie gramatyki.

Głównymi zaletami ANTLR są :

- Elastyczność - po zdefiniowaniu gramatyki ANTLR może zbudować kilka parserów w rożnych językach
- Logika przetwarzania jest zawarta w kodzie parsera, tzn że każda reguła produkcji gramatyki jest reprezentowana przez funkcję w kodzie parsera.
- Generuje parsery 'top-down' co pozwala na użycie bardziej ogólnych gramatyk
- GUI, dzięki któremu debuggowanie jest łatwe i zrozumiałe.
- Lexer, parser, drzewo przejść mogą być wygenerowane tylko z samej gramatyki
- ANTLR Mega Tutorial

4 Mechanika tłumaczenia

ANTLR z gramatyki generuje lexer, który bierze pojedyncze znaki i transformuje je w tokeny. Z kolei tokeny są atomami, które parser wykorzystuje do stworzenia logicznej struktury. ANTLR tworzy również klasę Listener, która nasłuchuje drzewo przejść. Na podstawie drzewa znana nam jest struktura pliku wejściowego w języku Python, co znacznie ułatwia implementację samego Listenera.

W Listenerze zaimplementowaliśmy funkcjonalności dla wszystkich reguł, które występują w kolejności z drzewa.

Na samym początku tworzymy plik wyjściowy. Nasłuchująca klasa wywołuje funkcję dla odpowiedniego wyrażenia . Funkcja ta tworzy w języku Java wyrażenie odpowiadające wyrażeniu w języku Python oraz kolejno konkatenuje je w całość.

Dodatkowo przetłumaczony plik wyjściowy dekorowany jest w statyczną klasę z funkcją 'execute', ta z kolei wykonywana jest w głównej klasie 'Main'.

5 Instrukcja instalacji

Projekt znajduje się na git hubie pod linkiem : https://github.com/grzepucz/Translator

Aby go uruchomić trzeba pobrać repozytorium i z poziomu konsoli uruchomić komendą: „mvn clean install”