Fakultät IV – Elektrotechnik und Informatik
MPGI 1 WS 2013/14
Pepper / Höger / Reicherdt / Zuber / Tutoren

Aufgabenblatt 12

letzte Aktualisierung: 16. 01, 10:23

Ausgabe: 16.01.2014
Abgabe: 23./24.01.2014

Thema: *Grammatiken, Scanner, Parser*

Achtung: In diesem Aufgabenblatt erwarten wir von euch ein **Testmodul**, das Testfunktionen enthält. Es soll **alle** eure Module der Hausaufgabe importieren und deren Funktionen mit **geeigneten** Testwerten füttern und **nachvollziehbare** Ausgaben liefern.

1. Aufgabe: Grammatiken

Im Rahmen dieses Tutoriums soll ein Programm zur Auswertung von aussagenlogischen Formeln entwickelt werden. Hierzu legen wir anhand einer Grammatik fest, welche Eingaben unser Programm als aussagenlogische Formeln akzeptieren soll und wie solche Eingaben für die Auswertung zerlegt werden müssen.

1.1. (Tut) Bestandteile Überlegt zunächst, aus welchen Grundbausteinen eine aussagenlogische Formel besteht. Beschränkt euch hierbei auf Konjunktionen, Disjunktionen und Negationen.

1.2. (Tut) Struktur Notiert nun eine Grammatik, mit der sich beliebige Formeln aus den Grundbausteinen erzeugen lassen. Beachtet dabei, dass Opal nur den Standardzeichensatz versteht.

2. Aufgabe: Scanner

Mit einem Scanner soll die Eingabe erst in die Grundbausteine unserer Sprache (Lexeme) zerlegt und die so erzeugten Partitionierung anschließend in sog. Token übersetzt werden. In den Vorgaben findet ihr dazu die Struktur `Token`, die einen passenden Datentyp `token` und eine Backtickfunktion zur Verfügung stellt:

```
DATA token == or and not true false open close
              error
```

Das Token `error` bietet dabei eine rudimentäre Möglichkeit Fehler anzuzeigen.

2.1. (Tut) Partitionierung Überlegt euch verschiedene Strategien zur Zerlegung der Eingabeformel (vom Typ `denotation`) in eine Sequenz aus Lexemen. Wie erkennt man, um welches Lexem es sich handelt? Wie wird kontrolliert, ob die Eingabe nur erlaubte Lexeme enthält? Schaut euch dabei insbesondere die Struktur `String` in der Bibliotheca Opalica an.

Beispiel für gültige Eingabe/Ausgabe-Paare:

```
"(T and (F or T))"  =>  <(,T,and,(,F,or,T,),)>
") T and or  ("     =>  <),T,and,or,(>
```

2.2. (Tut) Übersetzung Wie übersetzt man nun die Sequenz aus Zeichenketten in Token? Z.B.

```
<")","T","and","or","(">  =>  <close,true,and,or,open>
```

2.3. (Tut) Scanner Ergänzt nun die Struktur `Logic` um eine Funktion `scan`, die als Eingabe eine `denotation` bekommt und als Ausgabe eine Sequenz von Token liefert:

```
FUN scan : denotation → seq[token]
```

3. Aufgabe: Parser

In dieser Aufgabe wollen wir die Sequenz von Token aus dem Scanner syntaktisch interpretieren, dafür soll nun der Parser entwickelt werden. Die Fehlerbehandlung soll dabei analog zum Scanner nicht im Vordergrund stehen und nur in rudimentärer Form vorhanden sein.

3.1. (Tut) Datenstruktur Entwickelt eine geeignete Datenstruktur, die die abstrakte Syntax von aussagenlogischen Formeln fasst. Wie lässt sich solch eine Datenstruktur aus der Grammatik ableiten?

3.2. (Tut) Parsing-Algorithmus Entwickelt nun einen Algorithmus, der eine Token-Sequenz in einen abstrakten Syntaxbaum umwandelt. Wie lauten die Funktionsdeklarationen der benötigten Funktionen und wie kann eine rudimentäre Fehlerbehandlung realisiert werden?

3.3. (Tut) Parser implementieren Nun soll der Parser in der Struktur `Parser` implementiert werden. Orientiert euch dabei an der in der ersten Aufgabe entwickelten *Grammatik*.

3.4. (Pflicht-Hausaufgabe) Erweiterung mit Implikation und Äquivalenz Erweitert den im Tutorium entwickelten Scanner und Parser für aussagenlogische Formeln um die Junktoren $\rightarrow$ (Implikation) und $\leftrightarrow$ (Äquivalenz). Welche Datenstruktruen in Scanner und Parser müssen hierzu angepasst werden?

4. Aufgabe: Auswertung von logischen Formeln

In dieser Aufgabe wollen wir den eingelesenen Term nun noch auswerten. Des Weiteren sollen alle einzelnen Schritte aus den vorhergehenden Aufgaben und die Auswertung zu einer Funktion zusammengefasst werden.

4.1. (Pflicht-Hausaufgabe) Auswertung Implementiert eine Funktion `evalExpr`, welche das Ergebnis des Parsers (also den Syntaxbaum) zu einem `bool` auswertet.

Hinweis: Implikation und Äquivalenz lassen sich auf Konjunktion, Disjunktion und Negation zurückführen:
$a \rightarrow b = \neg a \vee b$
$a \rightarrow b = \neg(a \wedge \neg b)$
$a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a)$

4.2. (Pflicht-Hausaufgabe) Kombination aller Schritte Jetzt sollen alle Schritte zusammen gefasst werden. Implementiert dafür die Funktion `eval`, welche eine `denotation` übergeben bekommt, diese einliest, auswertet und das Ergebnis ausgibt. Verwendet dabei die entwickelten Funktionen aus den vorherigen Aufgaben.