Prof. Dr. Arnd Poetzsch-Heffter
Dr.-Ing. Jan Schäfer

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 3: Übersetzer und sprachverarbeitende Werkzeuge (SS 2011)

Hand Out: 4. Mai 2011
Hand In: 11. Mai 2011

Aufgabe 1 Eindeutigkeit von Grammatiken

Welche der folgenden Grammatiken sind mehrdeutig? Welche Sprachen beschreiben sie? Begründen Sie Ihre Antwort.

a) $N = \{S\}$ $T = \{\mathbf{0}, \mathbf{1}\}$ $\Pi = \{S \to \mathbf{0}S\mathbf{1} \mid \mathbf{0}\mathbf{1}\}$
b) $N = \{S\}$ $T = \{\mathbf{a}, +, -\}$ $\Pi = \{S \to +SS \mid -SS \mid \mathbf{a}\}$
c) $N = \{S\}$ $T = \{(,)\}$ $\Pi = \{S \to S(S)S \mid \epsilon\}$
d) $N = \{S\}$ $T = \{\mathbf{a}, \mathbf{b}\}$ $\Pi = \{S \to \mathbf{a}S\mathbf{a}S \mid \mathbf{b}S\mathbf{a}S \mid \epsilon\}$
e) $N = \{S\}$ $T = \{\mathbf{a}, +, *, (,)\}$ $\Pi = \{S \to \mathbf{a} \mid S+S \mid SS \mid S* \mid (S)\}$

Aufgabe 2 Konstruktion von Grammatiken

Erstellen Sie eine Grammatik für die Sprache, die alle Teilmengen der Java-Modifier **`public`**, **`final`** und **`static`** in allen möglichen Reihenfolgen akzeptiert. Wie würden Sie diese Aufgabe in einem realistischen Compiler für eine größere Menge an Modifiern lösen?

Aufgabe 3 Rekursiver Abstieg

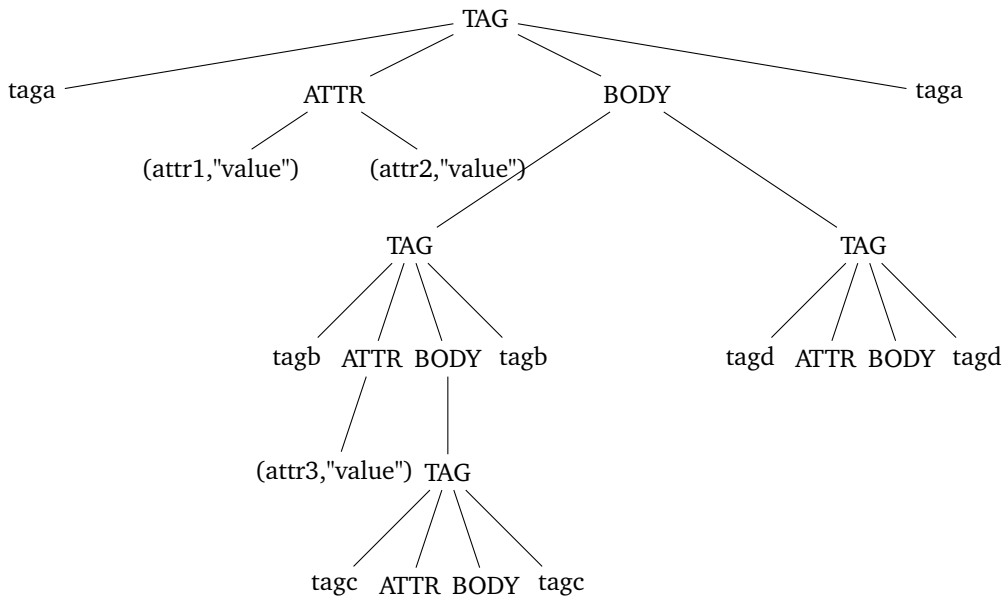
a) Erstellen Sie eine Grammatik für vereinfachte XML-Dokumente die für einen Recursive-Descent-Parser geeignet ist.

   Diese Dokumente bestehen aus beliebig verschachtelten Tags, mit beliebig vielen Attributen, d.h. Name-Wert-Paaren. Es sind beliebige Tag- und Attributbezeichner erlaubt. Text zwischen Tags, wie z. B. der Text eines XHTML-Dokuments ist **nicht** erlaubt. Whitespace zwischen Tags wird ignoriert. XML-Header wie z. B. `<?xml version="1.0" encoding="UTF-8"?>`, sind nicht in der Sprache enthalten. Die Grammatik muss **nicht** sicherstellen, dass die Bezeichner von öffnenden und schließenden Tags korrekt gepaart sind.

   Die Grammatik sollte die Baumstrukturierung von XML widerspiegeln, d. h. es sollte sinnvoll möglich sein, mit einem Parser für Ihre Grammatik, den in der Abbildung dargestellten Baum aufzubauen.

   Hinweis: Überlegen Sie sorgfältig, welche Terminalsymbole Sie wählen.

```
<taga attr1="value" attr2="value">
  <tagb attr3="value">
    <tagc/>
  </tagb>
  <tagd/>
</taga>
```



b) Implementieren Sie den Parser nach dem Schema der Vorlesung (ohne Parser-Generator). Verwenden Sie zum Scannen JFlex. Ihr Programm soll eine Datei einlesen und die Baumstruktur als Text ausgeben, so dass man nachvollziehen kann, dass korrekt geparst wurde.

Hinweis zur Abgabe: Die Abgabe muss zusätzlich zu den Source-Dateien auch ein Build-Skript enthalten, dass automatisch alle nötigen Dateien generiert, übersetzt und testet. Verwenden Sie dazu entweder ein Build-Tool (z. B. Ant oder Make) oder ein Unix-Shell-Script. Das Build-Skript muss unter Linux funktionieren, wobei Sie davon ausgehen können, dass `java` und `javac` ausführbar und im `PATH` sind und dass die `JFlex.jar`-Datei im `CLASSPATH` ist (bitte nicht in der Abgabe mitliefern). Erstellen Sie außerdem eine `README.TXT` Datei in der Sie erläutern wie das Build-Skript aufzurufen ist. Fügen Sie ebenfalls mindestens 6 Test-XML-Dateien zu Ihren Sourcen hinzu, die zum automatischen Testen verwendet werden. Die Test-Dateien sollten alle wesentliche Aspekte des Parsers überprüfen, wobei davon 3 positive Tests und 3 negative Tests sein sollten.

Aufgabe 4 (Optional) XML-Validierung

Erweitern Sie ihren Parser um eine (einfache) XML-Validierung. Sie sollte mindestens in der Lage sein, die korrekte Klammerung von Tags zu überprüfen. Geben Sie im Fehlerfall eine aussagekräftige Fehlermeldung auf der Konsole aus.