Bericht Sprachkonzepte

Autoren:

- Sascha Ivan
- Niklas Pelz
- Florian Halter

Aufgabe 1

Fragestellung:

- Schreiben Sie alle Programmiersprachen (im weitesten Sinne) auf, die Ihnen bisher im Studium begegnet sind. Überlegen Sie zu jeder Sprache, welche Sprachmittel sie für den Umgang mit hierarchischer Gliederung, Abstraktion, Redundanz und Abhängigkeit bietet. Wir diskutieren Ihre Überlegungen in der Übungsstunde.

Lösung:

- Übung 1

Aufgabe 2

Teil a)

Fragestellung:

- Schreiben Sie ein Java-Programm, das in einem String Formatspezifikationen gemäß java.util.Formatter findet. Erstellen Sie dazu mit der Syntax von java.util.regex.Pattern einen regulären Ausdruck für eine solche Formatspezifikation.

```
Beispieleingaben:
xxx %d yyy%n
xxx%1$d yyy
%1$-02.3dyyy
Beispielausgaben:
TEXT("xxx ")FORMAT("%d")TEXT(" yyy")FORMAT("%n")
TEXT("xxx")FORMAT("%1$d")TEXT(" yyy")
FORMAT("%1$-02.3d")TEXT("yyy")
```

Lösung:

- Uebung2.java

Teil b)

Fragestellung:

- Erkennen Sie mit ANTLR 4 Lexer-Regeln Zeitangaben im digitalen 12-Sunden-Format. Beachten Sie auch die alternativen Schreibweisen 12 midnight und 12 noon

Lösung:

- TimeSimple
- Time

Wir haben 2 Lösungen erstellt. In TimeSimple.g4 wird der eingelesene String als 1 Token mit Hilfe von Fragments erkannt. In Time.g4 wird der String in verschiedene Tokens unterteilt.

Ausführung:

- Ausführungsskript
    - Befehl: .\Uebung2.ps1 buildSimple timeBeispiel.txt
    - Befehl: .\Uebung2.ps1 build timeBeispiel.txt

Aufgabe 3

Teil a)

Fragestellung:

- Denken Sie sich eine kleine Sprache aus. Definieren Sie deren Vokabular mit einer ANTLR4 lexer grammar und deren Grammatik mit einer ANTLR4 parser grammar.

Hier haben wir uns für eine Sprache entschieden die bestimmte Charaktere aus einem Videospiel namens "League of Legends", ihre Items und ihren Score akzeptiert. Um dies zu ermöglichen haben wir erstmal ein sehr ausführliches Vokabular aufgebaut das jeden einzelnen Charakter + Item enthält:

Lexer Grammar:

- SaschLexer.g4

Um dieses Vokabular nun in unserem "Matchup" Format nutzen zu können benötigten wir natürlich auch eine Grammatik die beschreibt wie die Token aneinandergereiht sein dürfen:

Parser Grammar:

- SaschParser.g4

Anschließend sollte das Ganze noch getestet werden, dafür haben wir ein paar Beispiel Matchups in unsere txt File gepackt:

Beispiel:

- SaschBeispiel.txt

Und dann noch das Übungs Skript angepasst um entweder nur die Tokens in der Konsole ausgeben zu lassen, oder den Parse Tree mit dem TestRig anzeigen zu lassen.

Ausführung:

- Ausführungsskript
    - Befehl: ./Uebung3.ps1 tokens SaschBeispiel.txt
    - Befehl: ./Uebung3.ps1 gui SaschBeispiel.txt

Teil b)

- Definieren Sie mit Java-Klassen die abstrakte Syntax Ihrer Sprache aus a) und schreiben Sie ein Java-Programm, das den ANTLR4 Parse Tree in einen AST überführt.

Aufgrund unserer Struktur hat es Sinn ergeben unsere Syntax in 3 Java Klassen aufzuteilen.

Die Side stellt dabei einen Charakter mit seinen Items und dem Score dar:

- Side.java

Matchup ist dann eine Side gegen die andere:

- Matchup.java

Und schließlich noch das Game, welches alle Matchups zusammenführt:

- Game.java

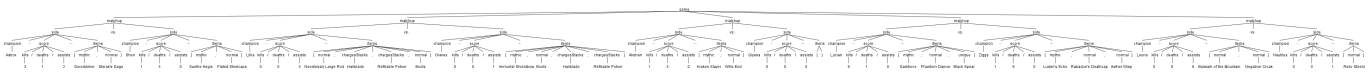
Um all diese Klassen nun in den AST zu überführen haben wir uns an dem ExprBuilder von Herrn Drachenfels orientiert, jedoch etwas vereinfacht und an unseren Fall angepasst:

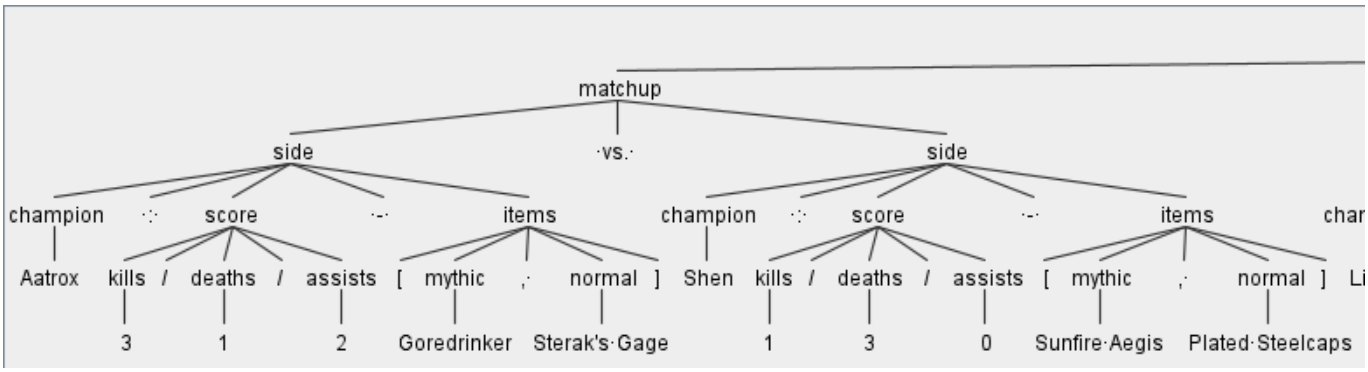
- ExprBuilder.java

Um das Ganze dann auszuführen haben wir die main Methode und unser Uebung3 Skript angepasst:

- ExprToAst.java
- Ausführungsskript
    - Befehl: ./Uebung3.ps1 ast SaschBeispiel.txt

Hier noch ein Beispiel zur Visualisierung der Struktur (Der gesamte Parse Tree ist etwas breit geworden):





Aufgabe 4

Teil a)

- Lässt sich eine statische Semantik für Ihre abstrakte Syntax angeben?

Ja tatsächlich lässt sich sich eine statische Semantik angeben. Es darf nicht derselbe Charakter auf der linken und der rechten Side vorkommen.

- Erlaubt Ihre konkrete Syntax Formulierungen, die die statische Semantik verletzen?

Es ist über die Grammatik ganz einfach möglich diese Regel zu brechen, da es nirgends formuliert wurde. Der Aufwand wäre viel zu hoch, da es über 150 Charaktere gibt.

- Ergänzen Sie gegebenenfalls eine statische Semantikprüfung für Ihre Sprache.

Um diesen Fall abzudecken haben wir unsere Charaktere alle in ein HashSet gepackt. Da in einem HashSet jeder Eintrag unique ist wäre die Größe des Sets bei einem sich wiederholenden Charakter kleiner 10, die Anzahl der Champions die in einem richtigen Spiel vorhanden sein müssen.

- checkStaticSemantic()

Teil b)

- Programmieren Sie für Ihre eigene Sprache aus Aufgabe 3 mindestens eine dynamische Semantik.

Bei der dynamischen Semantik nutzen wir den Score um zu berechnen welche Seite mit wievielen Kills führt:

- checkDynamicSemantic()

Aufgabe 5

Teil a)

Fragestellung:

- Vervollständigen Sie das folgende Java-Programm, indem Sie die aufgerufenen Klassenmethoden ergänzen.

Lösung:

- Was an dem Java-Programm ist eindeutig prozeduraler Stil?

Die einzelnen Prozeduren liefern Ergebnisse als Rückgabewerte und sie sind in überschaubare Teile zerlegt.

- Aufgabe5_a.java

Teil b)

Fragestellung:

- Stellen Sie das Programm aus 5a mit Hilfe von java.util.streams und Lambdas auf einen funktionalen Stil um. Ihr Programm darf nach der Umstellung keine Schleifen und Verzweigungen mehr enthalten.

Lösung:

- Aufgabe5_b.java

Teil c)

Fragestellung:

- Vergleichen Sie die Laufzeiten der Programme aus 5a und 5b.

Lösung:

- Ausführungsskript
- Befehl: .\Uebung5.ps1 Test.txt
- Output:

```
Aufgabe 5a): result = 152 (1335 microsec)
Aufgabe 5b): result = 152 (21612 microsec)
```

- Streams sind im Gegensatz zu Loops mit unserem Workload langsamer.

Aufgabe 6

Teil a)

- Folie 25

[X,Y,Z] = [john,likes,fish].

X = john, Y = likes, Z = fish.

[X, Y|Z] = [john,likes,fish].

X = john, Y = likes, Z = [fish].

[cat] = [X|Y].

X = cat, Y = [].

[[the,Y]|Z] = [[X,hare],[is,here]].

Y = hare, Z = [is,here], X = the.

[white,horse] = [horse,X].

false.

[white|Q] = [P,horse].

Q = [horse], P = white.

- Folie 26

```
fak(0, 1).
fak(N, F):-
    N > 0,
    N1 is N - 1,
    fak(N1, F1),
    F is N \* F1.
```

- Folie 28

Die erste Anfrage steckt alle Elemente erstmal alle Elemente der Liste in Y, diese werden dann pro Iteration eins nach dem anderen nach X geschoben. Die zweite Anfrage endet in einem Stack Overflow, da das Programm in eine Endlosschleife gerät.

Teil b)

- Programmieren Sie ein Prädikat sum, das die Summe einer Liste von Zahlen berechnet.

```
sum([], 0).
sum([H|T], Sum) :-
    sum(T, Rest),
    Sum is H + Rest.
```

Teil c)

- Definieren Sie ein Prädikat verbindung, das beschreibt, ob zwischen zwei Städten nach einer gegebenen Abfahrtszeit eine Verbindung inklusive Umsteigen existiert.

Nach leichten Startschwierigkeiten hat Prolog ab hier für uns erst wirklich Sinn ergeben. Davor haben wir die Art und Weise wie Prolog mit Rekursion arbeitet nicht wirklich verstanden. Mithilfe des Start Ortes aus dem ersten "verbindung" Aufruf konnten wir alle Züge ermitteln die von dort aus überhaupt losfahren. Anschließend kommt die Überprüfung ob die Uhrzeit um die der Zug losfährt vor unserer Ankunftszeit ist. Danach konnten wir mithilfe der Rekursion den nächsten Umstieg ermitteln, so lange bis wir in die obere

Abfrage reinrutschen da wir am Endziel angekommen sind und den letzten Zug in unseren Reiseplan hinzufügen.

```
verbindung(Start, Zeit, End_Ziel, Reiseplan) :-
    zug(Start, Start_Zeit, End_Ziel, Ziel_Zeit),
    Start_Zeit >= Zeit,
    append([Start, Start_Zeit, End_Ziel, Ziel_Zeit], [], Reiseplan).

verbindung(Start, Zeit, End_Ziel, Reiseplan) :-
    zug(Start, Start_Zeit, Umstieg, Umstieg_Zeit),
    Start_Zeit >= Zeit,
    verbindung(Umstieg, Umstieg_Zeit, End_Ziel, New_Reiseplan),
    append([Start, Start_Zeit, Umstieg, Umstieg_Zeit], New_Reiseplan, Reiseplan).
```

Aufgabe 7

Aufgabenstellung:

- Implementieren Sie eine Java-Anwendung, die für beliebige Java-Klassen und -Interfaces eine HTML-Seite im Format der Beispieldatei aufgabe7.html (siehe Moodle-Kursseite) generiert. Leiten Sie dazu aus aufgabe7.html eine Stringtemplategroup-Datei aufgabe7.stg ab. Die Java-Anwendung soll die gewünschten voll qualifizierten Klassen- und Interfacenamen als Aufrufparameter bekommen und mit Hilfe der Templates die HTML-Darstellung erzeugen.

Lösung:

- Beispieldatei
- Generierung des Class Arrays und Übergabe an Stringtemplate mit Hilfe der Template Engine: Aufgabe7.java
- Stringtemplate: aufgabe7.stg
- Ausführungsskript
- Befehl: .\Uebung7.ps1
- Output: output.html

Aufgabe 8

Aufgabenstellung:

- Implementieren Sie eine kleine Anwendung mit einer Scriptsprache und analysieren Sie, welche typischen Eigenschaften einer Scriptsprache Sie dabei ausnutzen.

Lösung:

- Unser Python Skript: Übung8.py
- Wir haben die Feiertage Api verwendet und 2 Parameter übergeben. Einmal `"jahr": 2022` und `"nur_land": "BW"`. Die Rückgabe Daten haben wir dann in eine json Datei geschrieben und formatiert

mit Hilfe der json Bibliothek von Python. Anschließend haben wir die Keys der json Datei mit Hilfe von einem Regex Pattern durchsucht.

- Analyse von Eigenschaften einer Skriptsprache:
    - Einfache Syntax: Keine main, keine Klassen
    - Deklarationsfreie Syntax (dynamische Typisierung)
        - implizit deklarierte Variablen
            - Zeile 5: parameters (dict[str, Any])
            - Zeile 12: feiertage(list))
    - Zeile 22-26: Pattern Matching mit regulären Ausdrücken