Hochschule für Technik und Wirtschaft Dresden
Fakultät Informatik/Mathematik
Prof. Dr.-Ing. habil. D. Müller

1. Mai 2017

Beleg zur Lehrveranstaltung „Software Factories“ SS 2017
Bearbeitung in Zweiergruppen

Gegeben ist die folgende Grammatik G_1 zur Definition einer rudimentären *Prolog*-Syntax:

G_1 = (N_1, T_1, R_1, <prologdsl>)

mit

N_1 = { <prologdsl>, <program>, <exquery<, <query>, <clause>, <term>, <functor>,
<predicate>, <fact>, <rule>, <atom>, <list>, <non empty list>, <folge> }
T_1 = { ident, number, variable, [,], |, ., ,, (,), ?-, :- }

R_1 =
<prologdsl> → <program> <exquery>
<program> → <clause>$^+$
<exquery> → ?- <query> .
<query> → <predicate> (, <predicate>)*
<clause> → <fact> | <rule>
<fact> → <predicate> .
<rule> → <predicate> :- <query> .
<predicate> → <functor> (<term> (, <term>)*)
<functor> → ident
<term> → <atom> | <list>
<atom> → ident | number | variable
<list> → [] | <non empty list>
<non empty list> → [<folge>]
| [<atom> | <term>]
<folge> → <atom> (, <atom>)*

Die konkrete Syntax der variablen Terminalsymbole ist wie folgt festgelegt:
- ident entsprechen `ID`, müssen aber mit Kleinbuchstaben beginnen,
- variable sind einzelne Großbuchstaben, z. B. `A` | `B` | `L` | `R` | `N` | `U` | `V` | `W` | `X` | `Y` | `Z`,
- number entsprechen `INT`.

Die zu realisierenden Aufgaben enthalten teilweise jeweils m nummerierte Teilaufgaben, von denen jeweils nur eine pro Gruppe (Standard: Zweiergruppe) zu lösen ist. Die Nummer n der von Ihnen zu lösenden Teilaufgabe berechnen Sie aus den Bibliotheksnummern b_i der Gruppenmitglieder nach $n = \left(\sum_{\text{Student mit } b_i \text{ Mitglied in Gruppe}} b_i \right) \text{modulo } m + 1$.

1. Es ist in *Eclipse* ein *Xtext*-Projekt (Nummern aufsteigend sortiert im Namensschema) `de.htwdd.sf.beleg.s`<erste Bibliotheksnummer>s<zweite Bibliotheksnummer> zu erstellen, in dem ein Metamodell als EMF-Genmodel zur abstrakten Syntax der Grammatik G_1 zu erstellen ist (Generierung der Code-Infrastruktur). Es ist eine DSL mit der *Xtext*-Grammatiksprache zu definieren.

2. Führen Sie das *Xtext*-Projekt als *Eclipse*-Applikation aus und erzeugen Sie eine DSL-Instanz mittels des generierten syntaxgesteuerten Editors.

(1)
```
vater_von(paul,peter).
vater_von(peter,otto).
vater_von(peter,ines).
vater_von(otto,leo).
grossvater_von(X,Z):-vater_von(X,Y),vater_von(Y,Z).
?-grossvater_von(U,V).
```

(2)
```
ist_nachbar_von(schmidt,meier).
ist_nachbar_von(meier,mueller).
ist_nachbar_von(mueller,krause).
?-ist_nachbar_von(schmidt,N),ist_nachbar_von(N,mueller).
```

(3)
```
prefix([],X).
prefix([X|Y],[X|Z]):-prefix(Y,Z).
?-prefix(W,[1,2,3]).
```

(4)
```
suffix(X,X).
suffix(W,[X|Y]):-suffix(W,Y).
?-suffix(W,[1,2,3]).
```

(5)
```
append([],X,X).
append([X|Y],Z,[X|W]):-append(Y,Z,W).
?-append([a|Y],Z,[a,b,c]).
```

(6)
```
member(X,[X|Y]).
member(X,[Y|Z]):-member(X,Z).
?-member(b,[a,b,c,d]).
```

(7)
```
auf(a,1).
auf(1,2).
auf(b,3).
auf(c,4).
auf(4,5).
auf(5,6).
ueber(X,Y):-auf(X,Y).
ueber(X,Y):-auf(X,Z),ueber(Z,Y).
?-ueber(X,Y).
```

(8)
```
reverse([],Z,Z).
reverse([X|Y],Z,U):-reverse(Y,[X|Z],U).
?-reverse([a,b,c],[],Z).
```

Die Baumdarstellung der Instanz in der *Outline*-Ansicht soll auch die konkreten Terminal- und die Nichtterminalsymbole enthalten.

3. Aufgabe ist die Programmierung einer *Xtend*-Codegenerierung zur Erzeugung einer Datei `prolog_s`<erste Bibliotheksnummer>`s`<zweite Bibliotheksnummer>`.lsp`, die den dem *Prolog*-Programm entsprechenden *Scheme*-Quelltext enthält. Dazu ist die folgende Grammatik G_2 gegeben:
G_2 = (N_2, T_2, R_2, <prologdsl>)

mit

N_2 = { <prologdsl>, <program>, <query>, <clause>, <predicate>, <functor>, <atom>, <term>, <fact>, <rule>, <constant>, <list>, <non empty list> }
T_2 = { ident, number, variable, `(`, `)`, `prolog`, `quote`, `cons`}

<prologdsl> → (`prolog` (quote <program>) (`quote` <query>))
<program> → (<clause>$^+$)
<query> → (<predicate>$^+$)
<clause> → <fact> | <rule>
<fact> → (<predicate>)
R_2 = <rule> → (<predicate> <predicate>$^+$)
<predicate> → (<functor> <term> <term>*)
<functor> → ident
<term> → <atom> | <list>
<atom> → ident | number | variable
<list> → `()` | <non empty list>
<non empty list> → (`cons` <atom> <term>)

Die konkrete Syntax für die Terminalsymbole ident, number und variable entspreche der der Grammatik G_1.
Speziell für <folge> gilt die Transformationsvorschrift:
[<atom> (`,` <atom>)*]
→
`(cons` <atom> `())`
| `(cons` <atom> <rest>)
mit
<rest> → `()` | `(cons` <atom> <rest>)

Beispiel: Ergebnis der Transformation für Aufgabe 2 (1):

```
(prolog (quote (((vater_von paul peter))
((vater_von peter otto))
((vater_von peter ines))
((vater_von otto leo))
((grossvater_von X Z)(vater_von X Y)(vater_von Y Z))))
(quote ((grossvater_von U V))))
```

4. Der bei Aufgabe 3 generierte Zielcode ist in der *JamusScheme*-Umgebung zu testen (Projekt `de.jamus.scheme` unter `\\iscad1\apps1 - PRAKT\fritzsch\SF\workspaceSF` in *Eclipse* importieren). Zur Ausführung wird das generierte Programm in die Datei `src/lsp_progs/beleg_prolog.lsp` des Projektes kopiert. Als Dateiendesymbol ist zusätzlich das Zeichen `]` als letztes Zeichen im Text der Datei einzutragen. Vor dem Einlesen des *Prolog*-Programmes ist von *JamusScheme* zunächst der *Prolog*-Interpreter `prolog.lsp` einzulesen. Zum Ausführen von *JamusScheme* als *Java*-Applikation sind anzugeben:

Project: `de.jamus.scheme`
Main class: `de.jamus.scheme.UseScheme`
Program arguments: `src/lsp_progs/prolog.lsp`

Das erhaltene Resultat der Abarbeitung ist (manuell) mit dem erwarteten Resultat zu vergleichen. Es sind optional jeweils selbst noch weitere Anfragen zu testen.

5. Stellen Sie die entwickelten Plug-ins (und der entwickelten DSL-Instanz) auf einer *Web-Site* zum *Download* bereit. Nutzen Sie den *Eclipse*-Update-Manager, um die entwickelten Plug-ins zu installieren (Erproben der Plug-ins). Senden Sie schließlich eine *E-Mail* an muellerd@..., in der Sie mir die Adresse der *Update-Site* und das gewählte Dateinamensuffix mitteilen. Anschließend müssen Sie Ihre Belegarbeit noch in einem Abnahmegespräch verteidigen.

Viel Spaß und Erfolg!