Generating Software from Specifications WS 2013/14 - Solution 2

Solution for Exercise 3

You have used Eli's documentation on "Products and Parameters" to understand and exercise important Eli commands.

Solution for Exercise 4

The file `TryScanLsg.fw` contains an extended version of the specification given in this exercise. A token specification, a correct and an erroneous input file (macros) are added.

Solution for Exercise 5

1. You repeated slides 3.1 to 3.4f of Chapter 3 in PLaC and answered the following questions:
2. Describe the different roles of the concrete and the abstract syntax in a language specification:
   The concrete syntax defines the structure of source programs. It has to be unambigous to generate a parser from it. The parser checks whether a given token sequence can be derived from the start symbol of the concrete syntax.
   The abstract syntax defines the structure of the structure trees (abstract program trees). It may be ambigous. Whenever the parser reduces according to a production that is not a chain production, it creates a node for a corresponding production of the abstract syntax.
3. How are precedences of operators specified in a concrete syntax?
   Chain productions, like Expr ::= Fact, Fact ::= Term, Term ::= Operand introduce different levels of precedence. Operators of lower precedence, as +, are defined on higher levels, e.g.

   ```
   Expr ::= Expr '+' Fact.
   ```

4. Why is it acceptable that an abstract syntax is ambiguous? The parser is generated from the concrete Syntax, which has to be unambigous. Reductions made by the parser cause tree nodes to be created. No decision is made based on the productions of the abstract syntax.
5. Explain the strategy for grammar design given on slide PLaC-3.4aa using the grammar on slide GSS-2.5:
   The grammar on slide GSS-2.5 is probably designed by a top-down strategy according to PLaC-3.4aa: The nonterminals Calender, Entry, Date, DayNames, GeneralPattern, and event are decomposed and described in this order.
6. Consider the grammar on slide GSS-2.5. Find all productions that specify arbitrary long sequences of certain language constructs.
   A non-empty sequence of Entry:

   ```
   Calendar: Entry+ .
   ```

   A non-empty sequence of DayName separated by comma:

   ```
   DayNames: DayName /
             DayNames ',' DayName.
   ```

7. Describe in what sense the sequences are structurally different.
   Sequences of DayNames have comma as a separators between any two elements. Sequences of Entry do not have a separator.
8. What other forms of sequences are possible, but do not occur in this grammar?
   Sequences that may be empty, and have no seperators.
9. Find all productions in that grammar which specify that some language construct is optional.
   A GeneralPattern has an optional Modifier, and an Event has an optional When-construct.
10. Consider the grammar on slide GSS-2.5. Translate each production into an English sentence, which describes the meaning of the production quite formally. You may combine strongly related productions into one description.
    The grammar description is given below.

The grammar description:

```
Calendar: Entry+ .
```

A Calendar is a non-empty sequence of Entry.

```
Entry: Date Event.
```

An Entry consists of a Date and an Event.

```
Date: DayNum '.' MonNum '.' /
      MonNum '/' DayNum /
      DayNames / GeneralPattern.
```

A date has one of four forms: a number of a day and a number of a month, both followed by a dot, a number of a month and a number of a day separated by a slash, DayNames, or a GeneralPattern.

```
DayNum: Integer.
MonNum: Integer.
```

DayNum and MonNum are both integral numbers.

```
DayNames: DayName /
          DayNames ',' DayName.
```

DayNames are a non-empty sequence of DayName separated by comma.

```
DayName: Day.
```

DayName is a Day.

```
GeneralPattern: SimplePattern /
                SimplePattern Modifier.
```

A GeneralPattern is a SimplePattern followed by an optional Modifier.

```
SimplePattern: 'Weekday' / 'Weekend'.
```

A SimplePattern is either the keyword 'Weekday' or 'Weekend'.

```
Modifier: '+' DayNames / '-' DayNames.
```

A Modifier is either a '+' or a '-' followed by DayNames.

```
Event: When Description / Description.
```

An Event is a Description preceded by an optional When-construct.

```
When: Time / Time '-' Time.
```

A When-construct is either a Time or two Times forming a time span.