Generating Software from Specifications WS 2013/14 - Solution 4

Solution for Exercise 9

This task exercises to generate parsers using Eli. Each of the files `parsing1.fw`, `parsing2.fw`, and `parsing3.fw` causes the parser generator PGS to complain that the grammar is "not LALR(1)". Further analysis of the concrete syntax and the examples of derivations given by PGS, yield the following results:

1. `parsing1.fw`:
   The given grammar is ambiguous. In the following statement sequence

   ```
   a = b; while c do c = d; i = k;
   ```

   the assignment `i = k;` may be the second statement of the loop body or it may be the first statement after the while statement in the whole sequence of statements.

   The ambiguity can be resolved in two ways: Allow only a single statement as loop body. Or, the statement sequence of the loop body is closed by a newly introduced keyword. See file `parsing1Sol.fw` in the directory `blatt4/calendarSol`.
2. `parsing2.fw`:
   The grammar specifies a non-empty sequence of declarations followed by a non-empty sequence of statements. Consider an input like

   ```
   int (a); f (b); h (a); a = 42; g (b); c = a + 5;
   ```

   Here, `f (b);` may be reduced to a `Declaration` (where `f` is reduced to a `Type`) or to a `Call` and then to a `Statement`. Before the first assignment is accepted, it can not be decided whether such a construct is to be reduced to a `Declaration` or to a `Statement`. Hence, the grammar is ambiguous, what implies it is not LALR(1) and not LR(1).

   In order to generate a parser, the grammar can be modified, such that `Call`s are also allowed in a `Declaration` sequence. The semantic analysis has to determine whether it is a `Call` or a `Declaration`. That usually requires to enlargen the language of the grammar. The file `parsing2Sol.fw` in the directory `blatt4/calendarSol` shows a parsable grammar and an input example.
3. `parsing3.fw`:
   This grammar exhibits a problem caused by using EBNF constructs. The parsable information shows that the three EBNF constructs are expanded using the created nonterminals `G1`, `G2`, and `G3`. The `parsing3.fw:consyntax` shows how they are used in the grammar which is send to the parser generator. The two alternatives for `Declaration` both begin with a sequence of `Ident` separated by`,`. Unfortunately, different nonterminals are used for the expansion of equivalent EBNF constructs. It would need unbounded lookahead to distinguish whether `Ident` is to be reduced to `G2` or to `G3`.

   The problem vanishes, if only one description of the sequence of `Ident` is used in both alternatives. See file `parsing3Sol.fw` in the directory `blatt4/calendarSol`.

Solution for Exercise 10

You developed a concrete syntax for a little language for sequences of typed declarations. It allows for several forms of simple and composed types.

Precedence rules for type composition operators were explicitly required. Make sure that you applied the production schemes to specify precedences and associativity. They need to introduce chain productions, which can be eliminated in the abstract syntax by mapping specifications. See file `SignaturesSol.fw` in the directory `blatt4/calendarSol`.

Solution for Exercise 11

See file `Calendar4Sol.fw` in the directory `blatt4/calendarSol` for a solution of this exercise. The comments that refer to this exercise are marked. Some comments are also given here:

1. Rule mapping can unify date notations like `23.5.` and `5/23`; but it can not additionally unify the alternative for `May 1`, because the month is represented by a terminal here.
2. Test cases for time spans are added.
3. A check for correct time spans is specified.
4. ... and tested.
5. A check whether days in a year are correct with respect to the number of days in every month requires to use attributes of type int to propagate the numbers representing days and months one context up.
6. Checks for spans of days in a week, and for spans of days in a year are implemented corespondingly.
7. All checks are tested with correct and erroneous test cases.

Solution for Exercise 12

No solution is given here.