Generating Software from Specifications WS 2013/14 - Assignment 5

Published: Nov 13, 2013 -- Turn in until Monday Nov 25 12h

What to turn in: see Assignment 1 and Exercise 15 below!

Exercise 13 (Learn to specify computations in tree contexts)

The directory `blatt5/calendarChk` contains a file `CalendarCheck.fw` which specifies a calendar processor as worked with in previous exercises. Copy it into a new directory and use it as described below.

This exercise guides you through extensions of that specification to get familiar with computations in tree contexts. Consult Eli's documents "LIDO -- Computations in Trees" and "LIDO - Reference Manual" for that work.

Make sure that you know the rules of the abstract syntax. It is influenced by a mapping specification. You may derive it for your information by `:absyntax`); but do not use the generated rule names, they may change. For each single subtask, you need only to specify those rules (or symbols) which you want to associate computations to.

It is advisable to place each solution of the following subtasks into a separate FunnelWeb macro having a unique file name, like `CalCheck-i.lido`, to activate and deactivate variants of solutions without deleting their text.

1. Introduce an attribute `Date.dayNum` and insert computations such that the day number in the year is stored and printed, e.g. by the following call:

   ```
   printf ("Day number %d in line %d\n", Date.dayNum, LINE);
   ```

   Test your processor carefully, and keep the test cases in your specification.

2. Design computations in tree contexts such that the earliest date mentioned in the input is found and printed in the root context. For that purpose introduce an attribute name

   ```
   ATTR minDate: int;
   ```

   which can be used with any nonterminal if needed. Every subtree which may contain a `Date` has an attribute that holds the smallest day number in that subtree. Where two values meet, the smaller one is propagated upward. You may use definitions of C macros for minimum computation:

   ```
   #ifndef MIN
   #define MIN(a,b) ((a)<(b)?(a):(b))
   #define I400() (400)
   #endif
   ```

   Put it into a suitable .h file.

   Test your processor carefully, and keep the test cases in your specification.

3. Deactivate the solution of the previous subtask, e.g. by changing the name of the macro's file type into `.nolido`. Then solve the same subtask using a CONSTITUENTS-WITH construct. Lookup the meaning and the use of such remote attribute accesses in Eli's documentation.

   Be aware how useful this construct is for that computational pattern.

4. Extend your computations such that wherever the earliest date of the input occurs, it is printed together with the line number of the context. For that purpose propagate the minimum value from the root down to all occurrences of a `Date`.

   Test your processor carefully, and keep the test cases in your specification.

5. Deactivate the solution of the previous subtask. Design a variant solution for propagating the minimum Value down by using an INCLUDING construct.

Test your processor carefully, and keep the test cases in your specification.

6. In oder to try a design variant, deactivate the solutions of all previous subtasks, and fill a new `.lido` macro. Make your specification less redundant: Use Symbol computations wherever possible.

   Test your processor carefully, and keep the test cases in your specification.

Exercise 14 (Patterns for Computations in Trees)

The directory `blatt5/calendarChk` contains a specification file `Statements.fw` which specifies a little statement language. The specification is complete up to construction of abstract program trees. The following tasks exercise the specification of computations in trees as described on the slides GSS-3.2 to GSS-3.8. Write your solution of each of the following sub-tasks into a `.lido` macro prepared in the specification file.

1. Augment the specification by computations of an attribute for each `Statement` node, such that it contains a unique number. It is irrelevant in which order the statements are enumerated. Introduce a counter variable and a count function into the C module that is provided at the end of the specification file. At each statement node output its number and the line number:

   ```
   printf ("statement %d in line %d\n", Statement.num, LINE);
   ```

2. Deactivate the previous solution. Achieve the same effect by using a CHAIN (cf. GSS-3.8): The unique numbers of statements are to be increased in left-to-right depth-first order. Output the statement numbers independent of the CHAIN.
3. Augment the specification by computations that count the total number of occurrences of a `VarNameUse`, and output it in the root context. Use a CONSTITUENTS construct as described in GSS-3.5.
4. Deactivate the previous solution. Specify computations of the nesting depth of `Block`s, as described in GSS-3.6. In the context of a defining occurrence of a variable create output like

   ```
   variable x defined in line 12 on block nesting level 2
   ```

5. Augment the specification by output at defining and at applied occurrences of variable names

   ```
   x defined in line 12
   x used in line 21
   ```

   Use dependence patterns as in GSS-3.7 to guarantee that all printings at defining occurrences precede all printings at used occurrences.

Exercise 15 (Describe your project)

Write a brief proposal for the development of a domain specific language of your choice:

1. Describe the domain and the intended language briefly (1 to 2 paragraphs).
2. Give a few examples for intended inputs in that language.
3. Describe briefly what the generator for that language is going to generate.

Turn that description in by email until the date given at the begin of the assignment page. Be prepared to present or discuss your proposal during one of the next lectures.

Exercise 16 (Homework, reading: Names and properties)

The semantic analysis phase for DSLs usually has to solve name analysis on the base of simple scope rules, and store and check properties of entities. So, recover your knowledge on those topics by

- recalling slides PLaC-5.1 to 5.4, 5.11, 5.12, 6.4, and
- reading Section "Basic Scope Rules" in Eli’s manual on "Name Analysis".

Do not forget to take notes on what you did, what you learned, and which problems you encountered, and turn them in.