Generating Software from Specifications WS 2013/14 - Assignment 3

Published: Oct 30, 2013 -- Turn in until Nov 6 at 12h

What to turn in: See Assignment 1

Exercise 6 (Specify token notation and coding)

This execise continues the exploration of token specification started in Exercise 4 of Assignment 2. Make sure that you did that exercise at least up to point 5. For this exercise use a fresh copy of the specification from `TryScan.fw`.

Before you execute the following sequence of tasks open the Eli manual on "Lexical Analysis" and keep it at hand to access helpful information.

a) Test output for tokens:
When you introduce a new token specification you need to test it. For that purpose prepare sequences of correct and incorrect tokens and put them into the two FunnelWeb macros `in.ok` and `in.err` in the specification file.

You may use the following Eli command to execute the generated processor with the input files:

```
. +cmd=(TryScan.fw:exe) (TryScan.fw:fwGen/in.ok) :run
```

Or you may extract the files into your working directory by

```
TryScan.fw:fwGen/in.ok > .
```

and execute the processor outside Eli.

Make sure that the tests yield the desired results.

For debugging you may as well use Eli's monitoring tool Noosa, as described in Exercise 4.

b) Tokens for days of the week:
Introduce a new kind of tokens into your specification to identify the days of the week by three letter abbreviations: Mon, Tue, etc. You need to specify the token notation by a regular expression (see the "Lexical Analysis" manual). Introduce a production for the new non-literal token into the concrete syntax.

Finally study the coding function `mkDay` which is provided in the specification file. Check whether it fits to your token notation and how it encodes tokens. Attach the function to your token specification.

Genenrate the processor with the parameter `+printtokens` and test the accepted tokens. Why is the result not satisfactory? Change the order of the token specifications to get satisfactory results. Explain!

c) Token codes in tree contexts:
Token processors, like `myToken`and `mkDay`, compute attribute values which are propagated into the contexts of the abstract program tree where the tokens occur. The specification file contains a macro of type `.lido` with two rules of the abstract syntax. Their `COMPUTATION` parts may contain computations to solve semantic analysis tasks. Now, we use that facility to output token encodings. Activate the print statements by removing the comment parentheses. Remove the parameter `+printtokens` and check and explain the resulting output.

Add another rule which corresponds to the concrete production for the day token you inserted in the previous task. Add a print statement to output the encoding of day tokens.

Find suitable, predefined coding functions in the Eli documentation, and attach them to the specifications of Number and Ident. Check how the token code for identifiers now has changed. Compare the codes for several occurrences of the same identifer.

d) Canned token descriptions:
Find in the "Lexical Analysis" documentation the section on "Canned Descriptions". Select a canned description for integral numbers and replace it for the specification of the `Number` token. Check the regular expression that is defined by the canned description, and add test cases to your input if notations for integral numbers are extended by this modification.

e) Canned identifier token description:
Substitute the token description for `Ident` by a suitable canned description. Does it allow additional identifier notations? If so, adapt the test cases.

Insert another printf operation in RULE p1. It shall print the string of the identifier using `StringTable(Ident)`.

f) Comments:
Introduce C-style comments to you specification, i.e. comments that are bracketed by /* and */.

Introduce also line comments in the style of Java or C. For that purpose you may take the canned description of ADA_COMMENT as a hint. Be aware that the slash character / needs to be escaped in the notation of regular expressions.

Exercise 7 (Specify scanning and parsing for the calendar generator)

Copy the file Calendar.fw from the directory `blatt3/calendar`. It is a specification of the lexical analysis and the parsing task for a calendar generator. It is derived from the corresponding descriptions in the lecture material and in the GSS book.

The annotations in the file contain four little tasks marked by **TASK i**. Solve each task in the given order. After each step an executable calendar generator ought to be generated; finally you get a version that checks the input. Tree construction is not yet specified.

Exercise 8 (Grammar modifications)

This exercise is intended to provide experience in modifications on context-free grammars specifying the concrete syntax. Continue your work on the calendar specification of the previous exercise.

Each of the following tasks asks you to do a little modification on the specified grammar. Make sure that you do NOT change the language by the modification! Test the specified processors.

1. Define `DayNames` using an EBNF construct.
2. Modify the structure of `Entry` such that the time or time span is not combined with the `Description`, but with the `Date`.
3. Modify the structure of `Entry` such that `Date`, time or time span, and `Description` are direct components of `Entry`.
4. Simplify the productions for patterns of dates: Specify `Modifier` to be optional; then remove unnecessary productions.
5. Finally extend the language such that also spans of dates can be described, like "20.8. - 5.9." or "Tue-Thu" but not "Tue-4.5.".