Generating Software from Specifications WS 2013/14 - Assignment 9

Published: Dec 18, 2013 -- Turn in until Jan 6, 2014 at 12h

What to turn in: see Assignment 1

Exercise 24 (Name analysis for qualified names)

In directory `blatt9/task` you find a file `QualNames.fw` which specifies a little language for structure type definitions, and field accesses using qualified names. The following tasks are to be solved:

1. The specification fragments are NOT explained. Read and understand the specification of name analysis and type analysis, and add explaining text to the specification.
2. A processor can be generated from that specification. However, it does not work as expected. Compare its results at least for the given examples to the expected results. The author of this specification has failed to specify necessary preconditions explicily for several computations. Add those necessary dependences, and test the corrected analysis.

Exercise 25 (Project: Requirements for semantic analysis)

Make sure that you have a complete set of written requirements for the semantic analysis of your language processor. In particular:

- Describe the use of names in your language.
- Where are defining and applied occurrences of identifiers?
- Has your language nested scopes or qualified names?
- Which violations have to be checked for name analysis?
- Describe properties of entities in your language, that need to be computed and checked.
- Which violations of properties have to be checked?

Exercise 26 (Project: Develop a test suite)

Work on your project and develop a complete test suite for all analysis phases:

Every error that may occur on the lexical level has to be exhibited.

The test suite has to contain syntactically correct examples, such that every language construct occurs. A few syntactically erroneous constructs are sufficient, because the parser is generated.

Elaborate carefully every situation where a syntactically correct progam in your language may violate any rule of static semantics (see previous exercise). For each such case

1. prepare an example which exhibits that violation;
2. state the rule that is violated, and explain the violation;
3. identify the context in the abstract syntax where the error message should be given;
4. what information is needed in that context to decide whether the violation occurs? Where is that information accessible?

Check whether you have covered every aspect of semantic analysis for your language. Keep the document ready to guide your implementation. Copy the directory `Test` with its sub-directories from `blatt9/task` into your project directory. Put the correct test cases into `Test/conform` and the erroneous test cases into `Test/deviate`.

The sub-directories of `Test` contain a script each, which runs your language processor on each test case. For the conforming cases it checks that no messages are given. For the deviating cases it compares the given messages to a file that contains the expected messages. Adapt these scripts such that they call your executable language processor correctly. Prepare for each deviating test case a file containing the expected error messages.