Project proposal
Master Informatique

An Eclipse environment for verified graph transformations

Projet Long, Enseeiht

Scientific Context

Within the Climt project[1], we are working on graph transformations, and in particular the verification of graph transformations.

Graph transformations have wide-spread applications, such as pointer-manipulating programs and model transformations in languages such as UML. They are also interesting for graph databases, such as those represented with RDF schemas, and the link structure of the World-Wide Web.

We are currently developing an imperative language for graph transformations, and a Hoare-style program logic for reasoning about programs of this language. The logic is a variant of a Description Logic.

Planned work

Global aim: We are interested in a programming environment integrated into Eclipse that permits to

- write transformations in our graph transformation language (GTL) in textual format, with syntax highlighting, auto-completion and dynamic type checking;
- perform dynamic verification of program correctness using the program logic;
- in the case of failed proofs, display a graphical counter model.

The resulting programming environment should become an easy to install Eclipse plugin.

Constraints: The code to be developed is not entirely stand-alone, but has to fit into an existing context. This induces the following constraints:

- *Existing code base:* The Eclipse plugin will be the front-end of an already existing back-end: Major parts of the GTL (syntax and semantics, Hoare logic) have been

[1] www.irit.fr/~Martin.Strecker/CLIMT/

formalized in the Isabelle[2] proof assistant, from which Scala[3] code is extracted that has to be interfaced with the front-end. The Description Logic prover, even though not generated from Isabelle, will be provided as a Scala class. All this motivates Scala as programming language for this project.

- *Design for change:* The GTL and program logic is the object of active research and will undergo heavy changes in the future (even though we will keep the language stable for the purposes of this project). The environment to be designed here therefore has to be easily modifiable.
- *Efficiency:* Program correctness proofs are time consuming. Special attention has to be paid to trigger new proof attempts only at clearly identified moments, and to localize the proof effort (for example to the innermost loop invariant for nested loops).

Prerequisites and interests

The project participants

- should have a sound understanding of programming language and compiler technology (parsing, type checking, language transformations).
- should have experience with OO (Java) and functional programming (Ocaml, Haskell). Under these conditions, they will quickly pick up a working knowledge of Scala.
- need not have prior knowledge of graph transformations. The notions are natural and are quickly acquired.
- need not have an in-depth knowledge of logic, proof search procedures or interactive theorem provers, even though an approximate understanding is helpful.

Administrative Context

For further inquiries, do not hesitate to contact Martin Strecker[4].

Please consult `http://www.irit.fr/~Martin.Strecker/Teaching/Master/` for Master project proposals in continuation of this “projet long”.

[2]http://isabelle.informatik.tu-muenchen.de/
[3]http://www.scala-lang.org/
[4]http://www.irit.fr/~Martin.Strecker/