A Framework for Proof Refactoring

Dominik Dietrich[1] and Iain Whiteside[2]

[1] Cyber-Physical Systems, DFKI Bremen, Germany

[2] School of Informatics, The University of Edinburgh

Abstract. We present a concrete framework for refactoring formal proof developments in a *generic*, *formal*, and *declarative* way based on graph rewriting and bidirectional transformation. Our approach uses a language-independent graph meta-model to represent proof developments in a generic way. We use graph rewriting to enrich the meta-model with dependency information and to perform refactorings; finally, our transformation model allows us to regain the modified syntax.

1 Quickstart

In this chapter we'll build a new system from scratch and perform a refactoring for an example that is shipped with the source code.

1.1 Download and Installation

The easiest way to use the system is to download the Binary `Refactoring.jar` from our website http://hompages.inf.ed.ac.uk/s0569509/refactoring.html. The system is shipped with

- an executable jar file `Refactoring.jar`
- several proof scripts that can be refactored using the tool in the examples folder.
- a meta-model in the directory `graphmodel`, which includes the appropriate GrGen files.

Make sure you have the following system requirements installed and available in the search path:

- Java 1.5 or higher
- GrGen 3.6 or higher (which can be downloaded from www.grgen.net). You will need to know the location of the GrShell binary.

```
wget http://homepages.inf.ed.ac.uk/s0569509/polar.tar.gz
tar xvfj polar.tar.gz
java -jar Refactoring.jar

java Refactor [options...] <scriptfile>
```

```
-I           : interactive (not yet implemented)
-abs FILE    : abstraction rules
-c FILE      : configuration file
-cmd VAL     : refactoring command
-ga          : create graph after refactoring
-gb          : create graph before refactoring
-gm FILE     : path to graph model files
-grshell VAL : GrShell executable
-o FILE      : output file
-p           : print theory after refactoring
-parser VAL  : parser
-printer VAL : pretty printer
-timeout N   : timeout in ms
-wdir FILE   : working directory

Example: java Refactor -I -abs FILE -c FILE -cmd VAL -ga -gb -
    gm FILE -grshell VAL -o FILE -p -parser VAL -printer VAL -
    timeout N -wdir FILE
```

1.2 Running Polar

To run the POLAR, you have to provide arguments that specify for example the path of the graph rewrite tool GrGen. It is also possible to provide a single configuration of the form `(parameter=value)*`, such as

```
parser = hiscriptparser.HiscriptSuite
timeout = 4000
wdir = /tmp
```

Table **??** describes the parameters of the tool.

2 Examples from Thesis

In this section, we describe how to run two of the example refactorings from the thesis. The *generalise tactic* refactoring cannot yet be run in the batch interface mode as we have a limitation on supplying parameters.

In particular, the `-cmd` parameter looks like

```
PLRenameLabel(10,10,\"a\")
```

The name of the refactoring rule is `PLRenameLabel`. The first two parameters are the line and column number of the label to be renamed. The rest of the parameters are the additional parameters to the refactoring rule. In this case there is only one more: the new label.

The limitation is that we can currently only reference one node. It is future work to improve the interface.

command	description
-I	Interactive Mode. GrGen is started in the interactive mode with the abstracted theory file loaded. Note that no theory analysis is executed. Type exit to exit the shell and to start the back propagation of the theory.
-abs	specifies the file that contains the abstraction rewrite rules. The abstraction rewrite rules are used to transform the original AST to a GrGen graph file.
-c	specifies the configuration file. The configuration file is a property file that can be used to store the standard configuration.
-cmd	specifies the refactoring command to be executed.
-ga	if set, the graph before performing the refactoring is stored under the name input-after.grs
-gb	if set, the graph before performing the refactoring is stored under the name input-before.grs
-grshell	specifies the path to the GrShell executable
-o FILE	specifies the file to which the output is written
-p	if set, the refactored theory is printed on the screen
-parser VAL	specifies the class that is used to parse the theory and to generate the initial AST
-printer VAL	specifies the class that is used to print the theory and from the refactored AST
-timeout N	specifies a timeout in ms to guarantee termination of the tool
-wdir FILE	specifies the working directory in which temporary files are created

Table 1. Program parameter

2.1 Renaming

The first example from the thesis is *renaming*. To perform a renaming on Hiscript, Polar can be invoked with the command:

```
java -jar Refactoring.jar  -abs abstraction/hiscriptabstraction.
    abs -grshell /usr/local/grgen/bin/GrShell -cmd "
    PLRenameLabel(4,12,\"Tryintro\")" examples/thesis.prfscr -
    printer hiscriptparser.hiscriptPrinterM -parser
    hiscriptparser.HiscriptSuite -timeout 15000 -gm graphmodel -
    o examples/thesisrenamed.prfscr
```

The 4,12 is used to identify the start of the name to be refactored. Tryintro is the new name. The refactored theory is called thesisrenamed.prfscr.

Similarly, to rename in the Omega language, the following command can be used:

```
java -jar Refactoring.jar -abs abstraction/omegaabstraction.abs
    -grshell /usr/local/grgen/bin/GrShell -cmd "PLRenameLabel
    (14,10,\"Tryintro\")" examples/thesis.omt -printer
    omegaparser.OmegaPrinterM -parser omegaparser.OmegaSuite -gm
     graphmodel -o examples/thesisrenamed.omt
```

2.2 Backward to forward

The second example from the thesis is *backwards to forwards*. To perform this refactoring, Polar can be invoked with the command:

```
java -jar Refactoring.jar  -abs abstraction/hiscriptabstraction.
    abs -grshell /usr/local/grgen/bin/GrShell -cmd "
    ConvertBackwardProof2ForwardHS(9,2)" examples/thesis.prfscr
    -printer hiscriptparser.hiscriptPrinterM -parser
    hiscriptparser.HiscriptSuite -timeout 15000 -gm graphmodel -
    o examples/thesisforward.prfscr
```

2.3 Generalise tactic

2.4 Debugging

If Polar provides an unexpected result or you want to understand the refactoring process in more detail, Polar provides an interactive mode which lets you perform commands over the command line step by step. Moreover, at each step of the process, the current stage of the transformation can be exported in a file.

```
public interface RefactorInterface {
    public LCommonTree getRefactorNode(int line, int column,
        GrGenGraph g);
    public Properties getTokenProps() throws IOException;
    public TokenStream getTokenStream();
    public LCommonTree getTheoryAST(File file) throws
        IOException, RecognitionException;
}
```

Listing 1.1. Interface that must be implemented by new parsers

Interactive Mode The `-I` option provides the possiblity to start an interactive mode. As a result, the program will abstract the initial proof script and export it to GrGen, but then start the GrGen's interactive mode. You can then manually invoke the theory analysis by typing `exec PLAnalysis`, view the graph in tool Ycomp by typing `exec show graph ycomp`, as well as performing any other command. Closing the shell using `exit` completes the cycle and invokes the projection phase and the generation of the resulting proof script. It is important to provide a correct timeout to avoid automatic termination of the tool after a specific time.

3 Extensions

The framework provided by Polar is generic. This means that the user can

- connect new proof languages to the system by providing a new parser and abstraction for the language
- add new graph models to the system and provide a mapping in that language via a corresponding abstraction

3.1 Adding a new language

To add a new proof language, the author must provide a new ANTLR parser that implements the interface shown in Listing **??**:

- `getRefactorNode` Given a line and a column, the function `getRefactorNode` computes the node in the abstracted AST that is handed to the refactoring command. Keep in mind that the actual node of the original AST might have been abstracted away; in this case, the best option usually is to provide the first predecessor node that is present in the abstracted AST.
- `getTokenProps` returns the ANTLR property file that defines the token types of the ANTLR parser. This information is used to identify the special nodes AT that introduce attribute nodes to the graph.
- `getTokenStream` returns the ANTLR token stream of the original file. This information is needed for the whitespace preserving pretty printing of the theory.

– `getTheoryAST` returns the theory AST for a given file, that is, performs the actual parsing.

It many cases, it should be possible to adapt the ANTLR grammars that are provided for the two example languages. Have a look at the grammar files that are in the directory `src/etc` of the sources.

3.2 Adding a new abstraction

Polar provides a simple abstraction language to define graph abstractions. The application of an abstraction φ to a theory AST t results in an abstraction $\varphi(t)$, which is directly exported to GrGen. Therefore, it is important that the abstraction is consistent with the graph meta model. The corresponding grammar is shown in Figure **??**. The reserved identifier `AT` is used to introduce attributes in the abstracted theory AST.

⟨*rules*⟩ ::= ⟨*rule*⟩*

⟨*rule*⟩ ::= expr '->' expr

⟨*expr*⟩ ::= ⟨*ID*⟩ | ID '*' | ⟨*STRING*⟩ | '(' ⟨*expr*⟩ ')'

Fig. 1. Grammar of abstraction files

An example of an abstraction is shown in Listing **??**.

```
(LFORMULA label term) -> (AT "name" label)
(THEORY name items*) -> (THEORY (AT "name" name) items*)
(THEOREM name formula proof) -> (THEOREM (AT "name" name) proof)
(FROM label*) -> (UREF (AT "name" label))*
(SHOW term proof) -> (GOAL proof)
```

Listing 1.2. example abstraction for Omega

3.3 Adding a new graph model

Graph meta-models define the meta-model for abstracted theory graphs. They are specified in form of a GrGen meta-model. An example is provided in the directory `graphmodel`.