Overview Exercises Rascal Tutorial PLDI 2015

Presenters

- Mark Hills: http://www.cs.ecu.edu/hillsma/
- Paul Klint: http://homepages.cwi.nl/~paulk/
- Jurgen Vinju: http://homepages.cwi.nl/~jurgenv/

Resources

General Rascal Resources

- Rascal home page: http://www.rascal-mpl.org
- Rascal at GitHub: https://github.com/cwi-swat/rascal
- The Rascal Tutor: http://tutor.rascal-mpl.org

Course Repository

- https://github.com/cwi-swat/rascal-course-pldi-2015.git

In this repository:

- `doc/Exercises.[md|pdf]` (this text)
- `doc/RascalCheatSheet.pdf` (overview of Rascal features)
- `presentations/*` (all course presentations)
- `src/*` (template code for the exercises)
- `lib/*` (library code used by the exercises)

Places to Find Rascal Examples

- The three exercise tracks described below.
- Rascal Recipes: http://tutor.rascal-mpl.org/Recipes/Recipes.html gives dozens of small examples.
- Rascal at Rosetta Code: http:rosettacode.org/wiki/Category:Rascal contains circa 50 Rosetta tasks with a solution in Rascal.

Three Exercise Tracks

You can select one of three exercise tracks (more details below):

- Hack Your Javascript
- Java Static Test Coverage
- PHP Analysis

Track 1: Hack Your Javascript

The *Hack Your Javascript* track shows how Rascal can be used to add various extensions to Javascript.

The *Hack Your Javascript* Track Illustrates

- Concrete pattern matching
- Desugaring and simple code generation
- Handling of variable bindings
- How a Rascal Eclipse plugin is organized

The Javascript exercises and all related sources and documentation can be found in a separate Github repo:

- https://github.com/cwi-swat/hack-your-javascript.git

Getting started with the *Hack Your Javascript* Track

1. Clone the above Github project
2. Setup is described in `doc/setup.pdf`
3. Exercises are described in `doc/HackYourLanguage_Exercises.pdf`
4. Import in Eclipse (File -> Import...)
5. In `src/Series1.rsc` and `src/Series2.rsc` you find skeleton answers to be filled in by you as well as tests to check your answer.

Track 2: Java Static Test Coverage

The *Java Static Test Coverage* Track Illustrates

- M3: reusable (*not* language independent) intermediate model for PL
- URI `loc` data-type for referencing and hyperlinking source code artifacts
- M3 query and reporting using relational calculus
- AST (abstract) pattern matching

Getting started with the *Java Static Test Coverage* Track

Get the code to analyze:

- Import the example project as follows: `File` menu, `Import`, `General`, `Existing Eclipse Project`, select `snakesAndLadders` (already provided in `data/snakesAndLadders`) to import from the `rascal-course-pldi-2015` clone
- start a console for the `rascal-course-pldi-2015` project

Type the following commands to get started:

- `import lang::java::jdt::m3::Core;`
- `import lang::java::m3::Core;`
- `m = createM3FromEclipseProject(|project://snakesAndLadders/|);`

Then import and edit the source file you will edit:

- `import java::JavaTestCoverage;`
- `:edit java::JavaTestCoverage`

The *TODO*s in the code comments at the bottom of the file point to places where you are expected to add to the existing functionality.

First try out some minor queries on the console REPL:

- `m@methodInvocation`
- `iprintln(m@containment)`
- Click on URIs in the console to jump to the source
- `import util::ValueUI;`
- `text(m@methodInvocation o m@methodOverrides<1,0>)`
- `problems(m)` and look at the Problems view to see the results.
- `stats(m)` and look at the printed test coverage report.

Track 3: PHP Analysis

The *PHP Analysis* Track Illustrates

- Using the PHP parser extract ASTs from individual files and entire PHP systems (optional, requires php)
- AST (abstract) pattern matching over the extracted ASTs
- Crafting queries to extract information into intermediate structures useful as a basis for richer analysis and analytics

Getting started with the *PHP Analysis* Track

All the code to analyze is included in the `data` directory, under `systems/WordPress/wordpress-4.2.2`. This is identical to the WordPress code available on the WordPress website under the release archive. Note that the WordPress installer has not been run on this copy, so the settings file created by the installer (`wp-settings.php`) is not present.

- WordPress website: https://wordpress.org/
- WordPress Release Archive: https://wordpress.org/download/release-archive/ There are two options for getting access to the WordPress ASTs:

- Build them from the WordPress source. This requires our custom fork of a PHP parser written in PHP, available on GitHub, and a recent version of PHP. Further instructions can be found on the PHP AiR GitHub project page. Note that this also requires some configuration for your environment; a sample configuration file is part of the PHP AiR distribution.

- PHP Parser GitHub Page: https://github.com/cwi-swat/PHP-Parser

- PHP AiR GitHub Page: https://github.com/cwi-swat/php-analysis/

- Use the existing serialized form of the ASTs. This is recommended for the tutorial, and removes the dependency on PHP.

Doing the Exercises

To use the serialized ASTs, run the following in a Rascal console:

- `import lang::php::ast::AbstractSyntax;`
- `import lang::php::ast::System;`
- `import php::Tutorial;`
- `wp = loadWordPress();`

If you have instead built the ASTs from source, you should have a statement like `wp = loadPHPFiles(wp422)` in your console, where `wp422` is the location of the WordPress 4.2.2 files (this is declared as a private variable in `php::Tutorial`, but you can copy it into the console or make it public in the module to use it yourself).

This will return a PHP `System`, which is a map from script locations to ASTs for scripts.

With this done, it is possible to start the exercises. These are in the `php::Exercises` module, with *TODO*s describing what needs to be done and documentation comments for each function. You should edit this module and import it to run your code (right click on the module and select `Import Current Module in Console`). The solutions to the exercises are in the `php::Solutions` module. Note that the exercises tend to increase in complexity, but this isn't absolute.