1 The Spoofax Language Workbench

The Spoofax language workbench [4] is a platform for the development of textual software languages with state-of-the-art IDE support. Spoofax provides a comprehensive environment that integrates syntax definition, name binding, type analysis, program transformation, code generation, and declarative specification of IDE components. It supports agile language design by allowing incremental, iterative development of languages and showing editors for the language under development alongside its definition.


Spoofax derives full-featured Eclipse editor plugins from such high-level language specifications. Spoofax itself is integrated into Eclipse by bootstrapping its language design DSLs.

Scalability. With Spoofax, large models or programs can be split over several files in a project. Spoofax supports incremental name and type analysis [7] of multiple files. This analysis is derived automatically from declarative name binding and type system specifications expressed in NaBL and TS. It consists of two phases. The first phase analyses lexical scopes and binding instances and creates deferred analysis tasks. A task captures a single name resolution or type analysis step. Tasks might depend on other tasks and are evaluated in the second phase. Incrementality is supported on file and task level. When a file changes, only this file is re-collected and only those tasks are re-evaluated, which are affected by the changes in the collected data. Spoofax does neither re-parse nor re-traverse unchanged files, even if they are affected by changes in other files.

Teamwork. With Spoofax, models and programs are stored in textual concrete syntax and can be managed in traditional revision control and source code management systems such as Subversion or Git. Teams can organise their work with languages developed in Spoofax in the same way they do with general-purpose programming languages. When a team member pulls in the changes of another
member, the source code management system will provide support for change tracking and merging. As a result, some files might be changed. Spoofax recognises changed files and performs an incremental analysis on these files. This will reveal any inconsistencies in the merged model or program.

2 QL and QLS Language Specification in Spoofax

For the previous edition of the language workbench competition, we developed language specifications for QL and QLS. We update these specifications and now use Spoofax’ new type system specification language TS to specify typing rules and constraints. We compile both languages to WebDSL [2], a domain specific language for web applications with rich data models, templates and access control. The WebDSL compiler generates Java web applications which use HTML, CSS, Javascript and MySQL.

QL. We specify the syntax of QL in SDF3 to obtain an editor with syntax checking, syntax highlighting and syntactic completion, and a formatter. Name binding rules and the type system are defined in NaBL and TS, providing us with incremental name and type analysis, constraint checking, reference resolution and semantic completion. To analyse dependencies and determinism, we specify additional analysis tasks in Stratego, using Spoofax’ task API for incremental analyses. Finally, code generation is implemented with rewrite rules with concrete object syntax in Stratego.

QLS. We specify QLS as an extension of QL, without requiring any modification to the QL specification. The analysis of QLS forms relies on name and type analysis of QL forms. We define additional name binding and typing rules for QLS constructs. The code generator relies on the dependency analysis of QL forms and extends the code generator of QL.

3 Workshop Presentation

We will start with a demonstration of a multifile QL/QLS project. We will focus on different teamwork scenarios as suggested in the LWC blog. For example, we will merge changes from another programmer into the project and show the errors that are shown by the IDE in response to changes. We will also discuss language features we added to specify missing but required questions.

Next, we will discuss some benchmarking results for the binary tree questionnaire. We will discuss how the sizes of changes, files and the overall project influence the response time of the editor and the overall compilation time.

Finally, we will give an impression how this behaviour is achieved. We will show how language designers can specify name and type analysis in declarative rules, without taking incrementality issues into account. We will sketch how Spoofax derives analysis tasks from these declarative specifications, how these tasks are evaluated, and how tasks support incrementality.
References