Towards Language-Parametric Refactorings

Philippe D. Misteli
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands
P.D.Misteli@student.tudelft.nl

ABSTRACT

A software refactoring is a program transformation that improves the structure of the code, while preserving its behavior. Most modern IDEs offer a number of automated refactorings as editor services. However, correctly implementing refactorings is notoriously complex, and these state-of-the-art implementations are known to be faulty and too restrictive.

Spoofax is a language workbench that allows language engineers to define languages through declarative specifications. When developing a new programming language, it is both difficult and time-consuming to implement automated refactoring transformations. The goal of this work is to implement sound language-parametric refactorings, which rely on an abstract program model built from the declarative specification of a language's static semantics.

CCS CONCEPTS

• **Software and its engineering → Compilers**; **Software maintenance tools**.

KEYWORDS

Refactoring, Renaming, Spoofax, Language-parametric

ACM Reference Format:
Philippe D. Misteli. 2020. Towards Language-Parametric Refactorings. In *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming (<Programming'20> Companion), March 23–26, 2020, Porto, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3397537.3398476

1 INTRODUCTION

A refactoring is a set of transformations on the source code of a program that changes its structure without affecting its observable behavior. The main goal of refactoring a program is to improve its code quality and thus its maintainability. The term was coined by Martin Fowler [3] in his well-known book and performing refactorings is an essential skill for every software developer to have.

Modern state-of-the-art IDEs, like IntelliJ or Visual Studio, offer automated refactorings as editor services. Refactoring using a tool should give a programmer the guarantee that the transformation is behaviour preserving and, therefore, should remove the need to test the program after the change. It is also significantly faster than doing it by hand, which allows the refactorings to be integrated more dynamically into the overall software development workflow. Despite the popularity of these built-in refactoring engines, they have been shown to be both flawed and too restrictive [2] [5].

The Spoofax Language Workbench [4] is a tool to develop domain-specific languages. The syntax and semantics of a language are defined through declarative specifications, which are written in meta-languages. The platform also offers features to generate common editor services, such as syntax highlighting and code completion. Having automated refactorings available out of the box, when developing a new DSL, would arguably be a welcome commodity for language engineers.

2 APPROACH

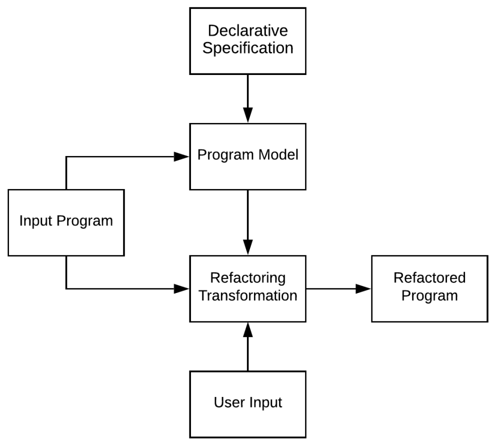

Figure 1: Architecture of Language-Parametric Refactoring

The goal of our research is to develop language-parametric refactoring transformations that guarantee preservation of a program's static and dynamic semantics. Language-parametric in this context means the transformations do not rely on specific syntax or semantics of the object language. Instead we rely on the safe-guarding of certain static semantic dependencies, such as name binding and data flow. Schäfer [7] introduced the notion of these dependencies in the context of refactorings and proved their usefulness by implementing common refactorings in Java. We aim to develop a

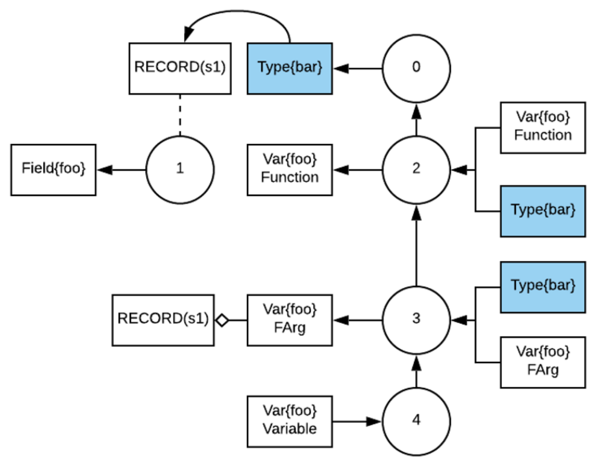

Figure 2: Scope Graph for Renaming Type foo

program model that concisely represents all the static semantic dependencies needed to assure a refactoring is behavior preserving, based on a language's declarative specification. The architecture of our proposed solution approach is depicted in Fig 1.

Spoofax allows a language's static semantics to be modeled through two integrated declarative meta-languages. Statix [10] is a constraint-language used to specify name binding and typing rules, which relies on scope graphs [6] to represent a program's name binding structure. FlowSpec [8] is a similar language used to construct the control-flow graph of a program and define data-flow analyses on its nodes.

We plan to implement an API using the Stratego [11] transformation language, which will allows us to implement the refactorings in the form of term rewrite rules. These rules will query the program model to find a terms associated static dependencies. As this program model can be built from any arbitrary language, the refactorings are language-parametric with regards to a language's static semantics specification.

```
1   let
2     type foo = {
3       foo : string
4     }
5     function foo (foo: foo) = (
6       let
7         var foo := foo.foo
8       in
9         print(foo)
10      end
11    )
12  in
13    foo(foo{foo = "foo"})
14  end
```

Listing 1: Renaming Example

As an example, consider the renaming of the type *foo* on line 2 in Listing 1. Based on the Statix specification, we can model the name binding structure of the program as the scope graph shown in Fig 2. Using the built-in name resolution algorithm, we can easily detect all the references to the type and then change the identifiers of the associated terms accordingly.

Our objective is to implement the Rename, Inline Function and Extract Function refactorings in such a language-parametric way. We chose these refactorings as they are amongst the most used ones by developers who publish their code on GitHub [9] and they rely on language features, specifically alphanumeric identifiers and function calls, that are common to most programming languages. To verify the refactorings, we plan to test the transformations on various languages for which Spoofax implementations exist, such as Tiger [1], Java and OCaml.

3 CONCLUSION

In this work we present a solution approach on how to develop language-parametric refactorings, by constructing an abstract program model from a language's declarative static semantics.

We plan to verify our approach by implementing the refactorings in the Spoofax language workbench and then test the transformations on languages developed in Spoofax.

REFERENCES

[1] Andrew W. Appel. 1998. *Modern Compiler Implementation: In ML* (1st ed.). Cambridge University Press, New York, NY, USA.

[2] Brett Daniel, Danny Dig, Kely Garcia, and Darko Marinov. 2007. Automated Testing of Refactoring Engines. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (Dubrovnik, Croatia) *(ESEC-FSE '07)*. Association for Computing Machinery, New York, NY, USA, 185–194. https://doi.org/10.1145/1287624.1287651

[3] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, Boston, MA, USA.

[4] Lennart C L Kats and Eelco Visser. 2010. *The Spoofax Language Workbench: Rules for Declarative Specification of Languages and IDEs.* Association for Computing Machinery. 444–463 pages. https://doi.org/10.1145/1932682.1869497

[5] Melina Mongiovi. 2016. Scaling Testing of Refactoring Engines. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 674–676. https://doi.org/10.1145/2889160.2891038

[6] Pierre Neron, Andrew Tolmach, Eelco Visser, and Guido Wachsmuth. 2015. A Theory of Name Resolution. In *Programming Languages and Systems*, Jan Vitek (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 205–231.

[7] Max Schaefer. 2010. *Specification, implementation and verification of refactorings.* Ph.D. Dissertation. University of Oxford, UK. http://ora.ox.ac.uk/objects/uuid:1a027679-1e2b-4fb5-a6ff-3270f15154a1

[8] Jeff Smits and Eelco Visser. 2017. FlowSpec: declarative dataflow analysis specification. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2017, Vancouver, BC, Canada, October 23-24, 2017.* 221–231. https://doi.org/10.1145/3136014.3136029

[9] Marco Tulio Valente. 2017. What are the most common refactoring operations performed by GitHub developers? https://medium.com/@aserg.ufmg/what-are-the-most-common-refactorings-performed-by-github-developers-896b0db96d9d.

[10] Hendrik van Antwerpen, Casper Bach Poulsen, Arjen Rouvoet, and Eelco Visser. 2018. Scopes as types. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–30. https://doi.org/10.1145/3276484

[11] Eelco Visser. 2004. *Program Transformation with Stratego/XT.* Springer Berlin Heidelberg, Berlin, Heidelberg, 216–238. https://doi.org/10.1007/978-3-540-25935-0_13