It's science. I'm not questioning it.

– Joshua Nichols

15 Summary

The motivation for this work has been to develop techniques and tools for expressing reusable, language-independent program analyses and transformations. It has been a goal that transformation programs should be reusable for language families, across language infrastructures and, when possible, across language paradigms. The following summarises the dissertation:

State of the Art First, a survey of existing software transformation systems is presented. The focus of the survey is on transformation languages and architectures. Attention is given to the program models, i.e. the facilities a transformation system has for representing and manipulating programs. The survey indicates that the program model is the central component that needs good abstraction facilities and a good abstract representation if one is to attain transformation reuse and language-independence.

Domain Analysis Next, an analysis of the program model is presented to find and describe the domain objects which must be abstracted over. This analysis suggests that the necessary abstractions behave in rather complicated ways: capturing them in traditional transformation libraries is not the best solution. It might be better to express the abstractions as new language features in the transformation language.

Design Subsequently, the above-noted abstractions are captured as abstract data types, and described using basic algebraic specifications. This formalises the abstractions and the rules governing them. Algebraic specifications work well as requirements for implementing the abstractions in transformation libraries. Algebraic axioms are a good source for deriving both optimisation rules and static checks for correct usage of the abstractions in the transformation library. This proved very useful in the implementation of the prototypes which were constructed for the Stratego transformation language. The abstractions are augmented with a domain-specific notation, making them full-fledged language features in Stratego. This provides a more convenient notation and raises the abstraction level for the programmer.

Implementation Finally, an extensible variant of Stratego called MetaStratego is presented. The prototypes of all the new abstractions were implemented on top of this extended system. The runtime of MetaStratego has some appealing features: it can operate on any tree- or graph-like program model, it executes on the Java Virtual Machine (JVM), it can be easily plugged into interactive development environments written in Java, and it can plug into compiler front-ends running on the JVM.

Proofs of Concept As proofs-of-concept, several prototypes were constructed, including:

- A modern, interactive development environment for (Meta)Stratego.
- A framework for interactive and automatic transformation and analysis of Java code.
- A code generator for axiom-based unit testing.
- A domain-specific aspect language for alerts.

The conclusion that this dissertation presents a significant step towards language independence for program transformation systems. It also indicates that there is more to be done before the goal is finally achieved.