13

Further Work

This chapter discusses possible future research directions based on some of the unresolved problems encountered during this research.

The aspect language (Chapter 5) may benefit from additional pointcuts that identify program locations based on program flow in the style of [AAC[+]05]. An additional extension would be to add support for dynamically weaving aspects into transformlets as these are loaded. For this to be possible, the aspect meta program must be maintained alongside main program at runtime so that it can be invoked when new transformlets are loaded. When using the aspect language for algorithm extension and adaptation of transformation skeletons, the resulting aspect programs sometimes become difficult to read and understand. There are patterns in these aspect-based adaptation schemes which may be distilled into new, higher-level language concepts. It is not yet clear exactly which patterns form the best foundations for a more general adaptation language.

The current graph language extension (Chapter 7) is minimal enough to capture cyclic graphs. This is sufficient for capturing the flow-based program models found in compilers, but may be insufficient for other applications. It may be useful to extend the language with a more general notation of graphs thus arriving at a more general concept of strategic graph rewriting. If this path is pursued, a graph visualisation plugin for Spoofax would be highly beneficial.

The program object model adapter technique (Chapter 4) has been applied for plugging term libraries, compilers and front-ends into the Stratego transformation system. Experimentation on higher-level programs, for example, ones based on software modelling approaches involving UML, may be fruitful. Together with the graph extension to Stratego, experiments with combining strategic rewriting with diagrammatic modelling approaches is possible.

The current selection of case studies demonstrates that the proposed techniques are applicable to realistic scenarios both for fully automatic and interactive program transformation. They do not cover the full range of transformations nor the full range of subject languages, however. Additional experiments involving additional subject languages are warranted.

Another area of future investigation is the pursuit of better support for language-independent transformations applicable to extensible subject languages. Consider a non-extensible subject language L. The syntax and semantics of L are known to the transformation developer when a transformation T is adapted to fit L. For an extensible language E, both the syntax and semantics may later be extended by programmers of E. These extensions may require additional cases to be added to T. The question of how these cases may be added requires additional research to answer. This topic is discussed more thoroughly in [Vis05b].

Additional mechanisms for the modularisation and adaptation of transformations is likely to be a fruitful topic of further study. Some experiments with a declarative "view" mechanism, somewhat related to that of Wadler [Wad87], have been conducted in the course of this research. These experiments are based on the concept of (generalised) signature morphisms. Initial experiments are encouragning, but significantly more work (and pages) is required before a conclusion can be reached.