Chapter 11

Conclusion

This document has endeavored to support the following thesis:

> **Thesis Statement:** *Logical frameworks based on a rewriting interpretation of substructural logics are suitable for modular specification of programming languages and formal reasoning about their properties.*

In the service of this thesis, we first developed a logical framework of substructural logical specifications (SLS) based on a rewriting interpretation of ordered linear lax logic (OL_3). Part I of the dissertation discusses the design of this logical framework, and in the process firmly establishes the elegant connection between two sets of techniques:

1. Canonical forms and hereditary substitution in a logical framework, on one hand, and
2. Focused derivations and cut admissibility in logic, on the other.

The broad outlines of this connection have been known for a decade, but this dissertation gives the first account of the connection that generalizes to all logical connectives. This connection allows the SLS framework to be presented as a syntactic refinement of focused ordered linear lax logic; the steps and traces of SLS, which provide its rewriting interpretation, are justified as partial proofs in focused ordered linear lax logic. SLS does move beyond the connection with focused logic due to the introduction of concurrent equality, which allows logically independent steps in a trace to be reordered; we conjecture that the resulting equivalence relation imposed on our logical framework is analogous to the one given by multifocusing in logic, but a full exposition of this connection is left for future work.

The SLS framework acts as a bridge between the world of logical frameworks, where deductive derivations are the principal objects of study, and the world of rewriting logic, where rewriting sequences that are similar to SLS traces are the principal objects of study. Part II of this dissertation discusses a number of ways of describing operational semantics specifications in SLS, using ordered resources to encode control structures, using mobile/linear resources to encode mutable state and concurrent communication, and using persistent resources to represent memoization and binding. Different styles of specification are connected to each other through systematic transformations on SLS specifications that we prove to be generally sound, a methodology named the *logical correspondence*, following Danvy et al.'s functional correspondence.

Most of the systematic transformations discussed in Chapter 6 and Chapter 7 – operationalization, defunctionalization, and destination-adding – are implemented in the SLS prototype implementation. Utilizing this implementation, we show in Appendix B that it is possible to fuse together a single coherent SLS specification of a MiniML language with concurrency, state, and communication using various different styles of specification, including natural semantics where appropriate.

This dissertation also discusses two different methodologies for formally reasoning about properties of operational semantics specifications in SLS. The program analysis methodology considered in Chapter 8 allows us to derive effectively executable abstractions of programming language semantics directly from operational semantics specifications in SLS. The methodology of progress, preservation, and type safety considered in Chapter 9 and Chapter 10 is presented as a natural extension of traditional "safety = progress + preservation" reasoning. In a sense, the work described in this document has pushed our ability to reason *formally* about properties of SLS specifications (and substructural operational semantics specifications in particular) some distance beyond our ability to *informally* reason about these specifications. An important direction for future work will be to move beyond the misleadingly-sequential language of SLS traces and develop a more user-friendly language for writing, talking, and thinking about traces in SLS, especially generative traces.

Bibliography