Chapter 9

Conclusions and Future Work

Throughout the research conducted in my thesis I proposed and validated solutions to unify two the domain of formal modeling and verification with the domain of software development and engineering. I discovered that in general researchers and professors with a more theoretical background are willing to adopt a mainstream programming language like Java or Scala into their research with more graduate students having acquired experience in programming during their studies. On the other hand industry is much more reserved to integrate formal methods and newly proposed language models, despite their strong academic backing, mainly due to the agile approach that focuses on getting products out quickly and making improvements on a continuous basis. As such my research slowly turned towards making formal languages, specifically ABS and its features, more approachable to software development and making it easier to learn and use by programmers. The research also focused on integrating ABS's features into the software development cycle. These included features for concurrent and functional programming, as well as features for modeling real-time systems with deadlines and resource provisioning for distributed systems.

This thesis formally describes a translation scheme together with a Java library for efficiently simulating the behaviour of ABS. The coroutine abstraction is a powerful programming technique in a software development context with a very strong research effort both in model proposals and implemented libraries. Having a scalable JVM library with support for coroutine emulation gives a basis for industrial adoption of the ABS language for component-based software engineering. It makes ABS a powerful extension of Java with support for formal verification, resource analysis and deadlock detection. The proposed library was stress-tested with a benchmarks tailored towards cooperative scheduling that show significant improvements of saving continuations including the call stack as data in memory instead of using a process-oriented approach by means native Java threads. With a better approach to the problem of using coroutines in programming comes the need to also compare it with state of the art solutions and therefore the library was tested against benchmarks for CPU-intensive applications to show that this feature does not have a negative impact on performance compared to existing actor libraries like the Akka Actor library.

There are however certain things that theoretical models cannot account for in software development. The Java language is a mainstream language with countless libraries and technologies developed around it that ABS is by no means intending to replace. This was the basis for extending the Java library to ASCOOP as a standalone library that directly offered ABS's functional and concurrent features in Scala. This extended library seamlessly integrates actors with futures and support for cooperative scheduling. The programming to interfaces paradigm enables common advantages of object-oriented programming including static type checking of message passing between actors as well as type hierarchies of actors. Tested with the Savina benchmarks the library is proves to be scalable in the number of actors, number of

messages communicated, and number of suspended sub-tasks. Comparisons with Akka and Scala actors show comparable performance while the simpler programming paradigm and statically typed messages lead to more ease of programming and less chance of runtime errors. The obtained reduction in the code is in part due to cooperative scheduling but also due to ASCOOP not requiring a context setup for actors and explicit build-up of messages and subsequent pattern matching on them.

The research towards unifying software development and formal models will continue in these two main directions covering several new aspects. At the translation level of ABS, the plan is to extend the ABS type-checker to verify the Foreign Language Interface constructs directly in ABS. ABS also requires the development of a debugger to enable profiling and visualization of concurrent programs. In both research directions there is still a need to further develop the research presented in Chapter 5 with respect to implementation of distributed actors. To allow systems of distributed actors, ASCOOP is extending the way completion of futures is propagated into a distributed push-based approach. Additionally, ASCOOP needs a mechanism to allow programmers to specify blocking tasks such that they can be scheduled on a separate executor; otherwise there is a risk that all available threads get blocked and normal CPU-intensive tasks get delayed. ASCOOP will also provide a more functional style of programming with futures as well as more integral error handling mechanisms.