# *Chapter 9*

# Conclusion

*Questions are never indiscreet.*
*Answers sometimes are.*

O. Wilde. An ideal husband.

## §9 Conclusion

In this chapter we briefly reiterate the material presented in the thesis, followed by a discussion of appropriate further work that could profitably be undertaken in this area. In Chapter 8 we have identified in detail the unresolved issues in the design of LSD, and in the user documentation in Appendix 3 we have described the known deficiencies in `am`, the implementation of the abstract defintive machine. We conclude by indicating possible future work which arises from this thesis

### §9.1 Summary

Chapter 1 gave an overview of existing approaches to sequential and parallel programming, with particular reference to the difficulty of parallelising sequential code. We then observed in Chapter 2 that state-transition systems can be effectively modelled by the use of definitions. A definitive notation is the language used to evaluate a state described by definitions and to perform transitions between states by redefinitions. A definitive program consists of a description of redefinitions which are to be performed autonomously. We described how definitions can in principle be performed in parallel without interference, and introduced the mechanism of guarded commands for describing when to execute redefinitions in parallel.

The abstract definitive machine (adm) was introduced as a vehicle for execution of definitive programs. The adm uses entities which can be instantiated and deleted to provide the ability to change according to the state the guarded commands which are in use. We considered how to deal with parameters, output, input, and singular states and invalid transitions.

In Chapter 3 we described the issues raised when implementing `am`, the simulator for the abstract definitive machine. We described the mechanisms for dealing with singular states and invalid transitions, how output and input are handled, and how the means chosen for storage of definitions may enable the von Neumann bottleneck problem to be avoided. The `rand()` function was introduced, and then an execution cycle in `am` was described. The chapter finished with two small example adm programs. User documentation for `am` is contained in Appendix 3 and maintenance documentation in Appendix 4.

Chapter 4 showed how to construct a program for the adm. It illustrated how synchronisation issues can be addressed by the use of the abstract definitive machine. Formal reasoning was applied to show that execution of the program cannot lead to interference. Appendix 5 contains an adm program to drive an existing package to produce an animation of the application. Limitations of the adm were described at the end of Chapter 4, as a motivation for the development of LSD in Chapter 5.

Chapter 5 introduced LSD, a specification language for describing a communicating system of processes acting concurrently. Agents, variables and protocols for action were described, followed by an explanation of how an LSD specification was to be interpreted as describing behaviour. Synchronisation between agents was then considered, with a distinction drawn between synchronisation by perception and synchronisation by assumption. A method was then supplied for transforming an LSD specification into a family of adm programs, each program representing a potential behaviour described by the LSD specification. Different adm programs were generated by changing the parameters of the transformation process, the parameters reflecting the extent to which agents are synchronised by perception or assumption. The example transformation used in Chapter 4 appears in Appendix 6, together with results from execution of the program.

Chapter 6 started with an LSD specification for a telephone system. This was then transformed using the techniques described in Chapter 5 to produce a family of adm programs, some of which are illustrated in Appendix 7.

Chapter 7 used an LSD specification to describe an educational application. The insight gained by considering the problem in these terms was then used to write an adm program for the application. The program and execution results are given in Appendix 8.

Chapter 8 contained an appraisal and comparison of the work described in the thesis with other approaches, and described some appropriate future work. The first part considered the adm. The use of definitions in the adm was discussed, and then the applicability of definitive programming to interaction, invariant maintenance and parallelism was considered. Definitive programming was then compared with other programming paradigms. Proposed extensions to the adm were then considered, describing windows, a superuser, and asynchronous execution of an adm program.

The second part of Chapter 8 discussed LSD and the transformation method described in Chapter 5. The use of definitions in LSD was considered, and then an appraisal was made of how our method can be used, covering specification simulation, analysis of results, interaction with the simulation, and simulation animation. We then described how our method of transforming an LSD specification into a family of adm programs can be used as the basis for a tool for the modelling and simulating of concurrent systems.

Appendix 1 describes current methods for describing concurrent behaviour, and Appendix 2 explains what is meant by the term "reactive system". Appendix 9 contains the source code for `am`, and Appendix 10 contains a glossary of the terms used in the thesis.

## §9.2 Further research

There are several major areas which need to be explored further than has been done in this thesis. Firstly, definitive programming needs to be fitted into a more formal framework. A suitable vehicle for this might be the state-transition graph of §2.1.1. This would allow the operational semantics of the adm to be formally specified.

Secondly, we have omitted consideration of traditional issues which arise in a concurrent framework, such as livelock, deadlock, starvation, fairness, and so on. These would need to be considered, both to allow a more complete evaluation of the properties of definitive system, and to allow a comparison with the approaches to representing and describing concurrent systems discussed in Appendix 1. Other issues we have not considered include programming methods, such as techniques to allow the implementation of recursive algorithms, and more sophisticated underlying and user-defined data types.

An important aspect of any research is the development of techniques to objectively evaluate its contribution to the field. To this end definitive programming should be applied to a large range of areas, so as to identify the advantages and weaknesses of this approach. In particular, it would be desirable to determine those applications to which definitive programming is particularly well suited, which will probably include those which involve the maintenance of relationships as a major computational overhead.

In Chapter 8 we have indicated several areas for future work: the formalisation of the place of the superuser

concept, the relating of the execution of the adm program to the LSD specification, techniques for the interactive selection of what is to be performed next in the simulation, the development of the menu-driven tool for modelling and simulating, and so on. All these areas are currently speculative, and would be appropriate issues to explore further.

We believe that the transformation from an LSD specification to a family of adm programs can be performed mechanically, although there is still an element of *ad hoc* additions to a transformed program. This is due in part to limitations of the simulator (i.e. the bugs described at the end of Chapter 3) and in part because we have not fully addressed every aspect of an LSD specification. The development of automatic transformation tools, for example by using a macro processor, might highlight those aspects which have not been fully resolved.

In conclusion we have presented a new programming paradigm called definitive (definition-based) programming and introduced the abstract definitive machine as a computational model. We have shown how the computational power of the abstract definitive machine can be used in conjunction with the cognitive expressiveness of the specification notation LSD for the effective modelling and simulation of concurrent systems.