Chapter 6

Conclusions, contributions and future work

This thesis has made a wide variety of contributions which can be categorised for the purposes of this conclusion into the following four themes:

- Conceptual machine models for EM;
- Empirical study of the EM tools and background scholarship;
- Identification of subtleties associated with dependency, and
- Strengthening connections with other work.

The next four subsections elaborate on the contributions made by this thesis to each of these themes. The final section hints at future application for this work.

6.1 Conceptual machine models for EM

Conceptual machine models of the ADM, the DAM machine and EDEN have been closely examined and related for the first time. This has involved significant original research and analysis as well as the refinement of pre-existing work. I have proposed a framework within which definitive systems implementations can be usefully understood. This framework is based on the ideas of Slade and encompasses a range of evaluation/storage strategies.

Slade's Abstract Definitive Machine (ADM) [Sla90] exemplifies a machine model that can be implemented using an "evaluate-at-use" strategy. This strategy is relatively simple to implement as only formulae need be stored. The emphasis is then on *actions* that cause evaluation, and dependency has a relatively low profile. I highlighted some ambiguities in the model of ADM execution as it has been described in the EM literature, introducing the concepts of major and minor state transitions in order to explain the ambiguity. These concepts are then used in later chapters.

Cartwright's Definitive Assembly Maintainer (DAM) machine [Car99] exemplifies an alternative "evaluate-at-redefinition" strategy. This strategy requires that values be stored in addition to formulae. The focus is then on *dependency*, which prescribes how change resulting from redefinitions should be propagated through the stored values in order to update the state. I highlighted some deficiencies in the DAM machine. Cartwright's Dependency Maintainer Model (DMM), on which the DAM machine implementation is based, formalises definitive notations as they currently exist. I make this more explicit by showing in greater detail the Low Level Definitive Notation (LLDN) concept assumed by Cartwright's DMM. My tri-box model presented in §5.3 is a proposal for a richer machine model to support dependency maintenance beyond what is possible with our current definitive notations. I have described the model using graphical examples. Questions remain about how to represent this model in a notation.

All other strategies for dependency maintenance can be regarded as blending evaluation-at-use with evaluation-at-redefinition. Such a strategy is illustrated in Y.W. Yung's EDEN [Yun90] which uses an "evaluate-at-use-when-necessary" strategy. The Eden language and EDEN implementation successfully integrate both actions *and* dependency, and, as a result, a wide variety of models can be constructed

using it. This thesis includes the first full exposition of the EDEN machine model, developed by deriving a pseudo-code for the scheduler algorithm and representing it in two types of diagram. This reveals many clever aspects of Yung's design. My exposition suggests ways of implementing higher order dependency using the existing Eden language, with a higher degree of confidence in its correctness. It also emerges that the EDEN algorithm also achieves an aim set out by Cartwright in [Car99]: that of minimising updates resulting from a block redefinition.

Despite these many positive aspects, the EDEN design is complex and sequential and it is difficult to see how to extend it to take advantage of the possibilities for concurrent update of dependencies. However, the integration of actions and dependency as distinct concepts within the Eden language allows me to show how definitions can be translated to actions, thereby making my notion of 'dependency-as-agency' more concrete. It then becomes clear that definitions are a form of triggered action with certain limitations. I name these actions *definition-agents*. This establishes the context for an initial analysis of concurrent definition maintenance in Chapter 5. The two primary issues are how to map definition-agents to the script graph and then how to synchronise (or sequentially, schedule) the actions of the definition-agents.

6.2 Empirical study of the EM tools and background scholarship

Building on some preliminary tests made by Cartwright, I compared the DAM machine with EDEN, paying particular attention to controlling differences of underlying platform. The tests show that the two definition maintainers actually have similar performance: any advantages that the !Donald application has over `tkeden` appears to be attributable to the different graphics subsystems used, not to the definition maintainer implementation.

Considering possible ways to evaluate a script graph, I constructed some minimal examples that can be used in black-box testing of any definition maintainer

in order to investigate the evaluation ordering that it implements. An analysis of the schedule ordering used by EDEN reveals that it evaluates breadth-first, oldest-to-newest. Definitions and actions appear to be scheduled using the same strategy. This observation feeds into the discussion of the distinction between dependency and agency mentioned above.

In various attempts to quantify aspects of the context of EDEN, I collected historical source code and analysed code growth. The resulting data shows continued growth since Y.W. Yung's original version of 1988, and also illustrates an early decision that I took as the maintainer of EDEN: to attempt to grow the tool through writing code in Eden rather than lower-level languages, in order to assist with portability, reliability and flexibility. I also compiled a table summarising over two years of data that I collected in order to measure usage of the tool. In the thesis, this illustrates the pattern of recent local usage.

These quantified aspects of historical context are mixed with careful reading of the various original sources, which have been reviewed in the light of our current understanding. Drawing together the writings exposes various inconsistencies due to the immaturity of our research area. In particular, there is a problem with the various terminologies used by different authors. The variation comes about for many reasons. It is partly due to continuing improvements in our understanding of the various topics. The broad scope of EM also causes difficulties, particularly for authors investigating applications of EM, who then have to deal with terminology both from EM and from their application area. This seems to have caused the meaning of some terms relating to EM machine models in particular to be inadvertently broadened beyond their original scope. I also suspect that limited accessibility of the original source material has made consistency difficult, and in order to conduct some of the research necessary for this thesis I have first made various efforts to improve the organisation and availability of our library. In this thesis, I have attempted to point out problems in some of the sources in particular in relation to machine models, and efforts are underway more generally within the group to define standard terms where we have developed them and to encourage their use.

6.3 Identification of subtleties associated with dependency

My introduction attempts to show the range of applicability of the dependency concept. An appreciation of the concept in full inevitably involves the intimately related concepts of observables and agency. Even if we restrict our attention to the domain of programming, various subtleties associated with dependency exist, which I attempt to highlight in this thesis.

Some of the issues have been identified before but have not received much attention. Meziani's [Mez87] concept of 'moding' for example is one of the early contributions that has had no effect on subsequent tool building. However it is an important concept that has significant bearing on problems we still encounter with composite structures such as definitive lists. I have reviewed moding problems in relation to broader problems of dependency maintenance for which this thesis offers partial solutions.

Some of these problems are considered in full for the first time in this thesis. The indirection involved in a symbolic reference is a problem that I have illustrated in the context of the DAM machine, where it is most clearly shown. I have also encountered the same problem in respect of several dependency structures that can be implemented only with difficulty in the current EDEN. The simplest problematic example illustrated in this thesis is that of the '`if`' Higher-Order Dependency (HOD). This is a particular problem for implementations that use doubly-linked pointers to indicate sources and triggers of symbols, as both EDEN and the DAM machine do. Implementations that use the evaluate-at-use strategy avoid this problem, as the focus is on actions rather than propagation of change. Since the essence of the '`if`' HOD problem lies in the detection and propagation of change rather than in evaluation, this problem has not been previously encountered in the Computer Science literature, where the emphasis is on implementing sequences of actions. The implementers of spreadsheet programs face the same problem with the INDIRECT function, and usually work around the problem rather than solving it in full. I have proposed a "tri-box model" that takes full account of the problem and provides a potential solution, but it may be hard to implement the model efficiently as it is currently conceived.

6.4 Strengthening connections with other work

Although the main body of this thesis contains only a few references to external literature, the thesis makes new and significant connections with other work. Technically, this is the first work to give a substantial treatment of the issue of concurrent dependency maintenance. Dependency requires a form of synchronisation that is reconfigured in varied ways depending upon factors including the topology of the current script graph and the location of change within it. A definition can be considered to be a new kind of synchronisation primitive that has interesting and useful properties: it gives a guarantee about the validity and explanation of the current value without the need for information hiding. Definitive scripts can be composed with other scripts, and can be connected to other procedural systems if sufficient knowledge of change is available.

In respect of the underlying philosophy for Empirical Modelling (that of generating "one experience that knows another"), this thesis emphasises the importance of tool building. Our concepts have been progressively refined and tested in application. However, the tools do not implement the concepts faithfully in many cases. In particular, I take the view in this thesis that our implementation techniques for tools should *embody* the concepts at as low a level as possible, rather than take the approach that the concepts are something to be implemented through abstraction. A computer tool that embodies the principles will provide a substrate to a modeller that can be compared to the pipes, tanks and fluid used by Phillips in constructing his machine (see §1.1). Such a substrate allows a modeller to interact with state in ways that are in no respect preconceived. A model constructed in such a substrate can then potentially be an extremely powerful metaphor for the modeller's understanding, giving the fullest possible support for interaction with meaningful state.

My experiments with DAM machine dependent pixels, the EDEN steering wheel and railway are the first to extend our concepts to broader aspects of computer-related technology. They demonstrate the more comprehensive view of computing which we intend EM to embrace.

6.5 Future work: dependable computing, psychology and foundations

I believe that dependency can in principle support more dependable computing. The prospects here are highlighted by the following quote from McCormick[1], which is also cited in the introduction.

> [Software] developers have always had to explain relationships within and between their systems. If they can explain those relationships with the simplicity and consistency demanded of other engineering disciplines, they will succeed. If not, it probably means that a dash for novelty has sprinted too far, too fast, and too soon.

Section §1.4 in the introduction looked briefly at circuits constructed from digital logic gates. They can be described in an uncontroversial diagrammatic form and easily extended or composed with other circuits. Such circuits can be specified using the simple and powerful Boolean algebra. These are illustrative of the simple, consistent, explicable relationships desired of software by McCormick.

It seems to me that our current foundations for computing make it exceptionally hard to construct programs that exhibit the simple, consistent and explicable relationships we require for dependable computing. This thesis contends that many of the problems of dependable computing stem from the pervasive reliance on sequences of action in the conception of software. It further proposes that indivisibility in state change — the central abstraction underlying circuits — can be the basis of a radical generalisation of circuit building.

Two aspects of circuits are particularly relevant to the research in this context:

- it is possible to exploit circuits in a wide range of applications (albeit sometimes courtesy of the von Neumann machine);
- circuits are rooted in a mature engineering discipline that provides guarantees about the simplicity and consistency of the relationships they establish.

Extensive previous work on Empirical Modelling has indicated that "generalised circuits" can potentially be exploited in just as wide a range of applications and that, moreover, it can address issues that are treated as separate concerns in the software-as-sequences-of-actions paradigm. The main contribution of this thesis is

[1] [McC] unpublished essay, quoted in [Lev95, p.509].

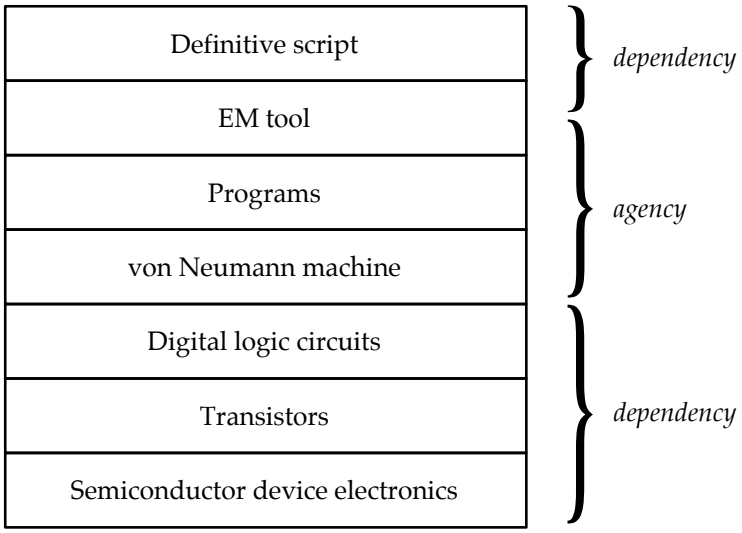

Figure 6.1: Dependency and agency in the current EM tool architecture

to explore the extent to which such generalised circuits can be rooted in a mature engineering discipline. In particular, the critiques of machine models show how the implementation and use of dependency may be amenable to engineering analysis.

There are two principal ways in which the research reported in this thesis can be extended towards the goal of supporting more dependable computing. Figure 6.1 depicts the way in which dependency in software and in circuits are related in the architecture of current EM tools. In the short term, it will be important to further the study of dependency-as-agency that was initiated in the analysis of EDEN in Chapter 4 and the DAM machine in Chapter 3. In the longer term, it may be possible to reduce the contribution that agency makes to the architecture. This contribution is already being eroded by the progressive re-implementation of EDEN within itself alluded to in Chapter 4. More radical ways of eliminating agency are suggested by the direct exploitation of the DAM machine video hardware described in §3.5.4.

But how will we know if we have succeeded in offering support for more dependable computing? We must move our studies beyond the anecdotal, perhaps first to case studies showing a correlation between the use of dependency and a more dependable product, and then ideally to a full true experiment demonstrating

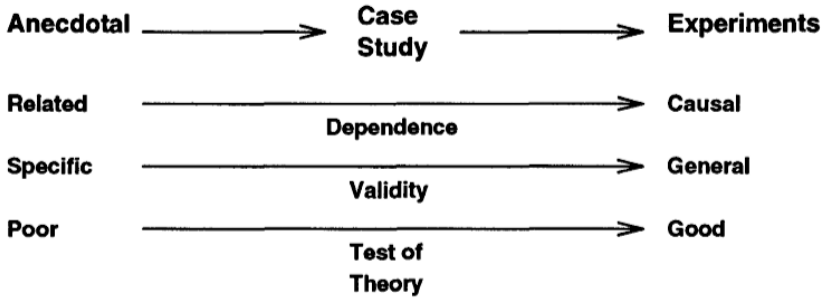

Figure 6.2: Spectrum of empirical work (from [VP95], © 1995 ACM — copy by permission of the Association for Computing Machinery)

causality and a constructive, testable theory [VP95] that explains why dependency enables us to produce a more dependable product. Figure 6.2 illustrates this range of possible empirical work.

It may be possible to construct such experiments in the classroom. In 2003-04, we set an assignment for 16 fourth year Computer Science MEng students taking our "Introduction to Empirical Modelling" module, firstly asking each of them to independently produce a `tkeden` model of a home central heating system. In a second (initially unseen) part to the assignment, we asked them to change their model in order to demonstrate scenarios involving heating system maintenance, malfunction and design changes (e.g. model the temporary disconnection of a radiator, a leaking boiler, the addition of a frost-protection feature). Our hypothesis is that a model which is largely constructed using dependency and based on an LSD account, as we recommend, should be easier to modify to these changed requirements than a simulation programmed largely from sequences of procedural actions without close and continuous connection of meaning to the referent. The assignment submissions taken as a whole appear to confirm this — there appears informally to be a correlation between the use of dependency and the ease of making changes in the second part of the assignment, but the data does not support a formal evaluation. Therefore we are still at the level of anecdotal evidence. Possibly a controlled experiment where one group of subjects would use the definitive and procedural features of EDEN, and another group only the procedural side, would be a good basis for such an evaluation. Continuing our exploration of the Cognitive Dimensions of definitive

notations, which we started in [BRWW01], also seems likely to lead to insights.

As with any contribution to human-computer interaction, it is necessary to produce hard, quantitative results, so that hypotheses may be confirmed or falsified. However, it seems that proper experimental design, analysis and reporting (by the standards of the psychology discipline) is still rare in the Computer Science field. Sheil [She81] states that:

> As practiced by computer science, the study of programming is an unholy mixture of mathematics (e.g. [Dij76]), literary criticism (e.g. [KP74]) and folklore (e.g. [FPB95]).

and recommends that:

> ... the methodological recommendations of computer science should be recognised as *empirically testable, psychological hypotheses*... a discipline of computer science has an obligation to validate these claims.

(It should be noted that Empirical Modelling is not a method, rather an "amethodical approach" as described by [TBT00], although it may be a methodology in the sense described by [Che99].)

There are many risks to the validity of experiments such as ours — one example is the effect of "demand characteristics" mentioned by Sheil, whereby participants modify their behaviour in response to their perception of the experimenter's desires. More recently, Kitchenham *et al* [KPP+02] state that in their view, "the standard of empirical software engineering research is poor", but go on to present what appear to be useful guidelines for such research.

This thesis has been an attempt to analyse and ground the concept of dependency and its implementation. The most formal description we now have is given by Chapter 5, which relates the concept to observation and agency. The ACM Task Force on the Core of Computer Science, chaired by Denning, reported [DCG+89] in 1989 that "The fundamental question underlying all of computing is, What can be (efficiently) automated[?]". The concept of dependency developed in this thesis is one that supports the indivisible change of meaningful state. To this extent, dependency offers an important and fundamental means of achieving automation. If programming — in a broad sense — refers to how all computer-based automation is to be managed, then dependency must play an essential role, complementary to that of logic, in the foundations of programming.