CONTRIBUTORS

H. BARRINGER
Department of Computer Science, The University, Manchester M13 9PL, UK

D. M. GABBAY
Department of Computing, Imperial College of Science and Technology, University of London, 180 Queen's Gate, London SW7 2BZ, UK

A. P. GALTON
Department of Computer Science, University of Exeter, Prince of Wales Road, Exeter EX4 4PT, UK

R. W. S. HALE
University of Cambridge Computing Laboratory, Corn Exchange Street, Cambridge CB2 3QG, UK

F. SADRI
Department of Computing, Imperial College of Science and Technology, University of London, 180 Queen's Gate, London SW7 2BZ, UK
FOREWORD

Computer science is the study of algorithms, programming languages for the expression of algorithms, and computer systems for the implementation of programming languages. An important activity in computer science is the invention, analysis and application of formally defined calculi, called logics, which are designed to specify, reason about, and represent these algorithms, programs and systems.

The articles in this book concern logics for reasoning about time, and illustrate their application in all three areas of computer science. For example, research on the logic of events is brought together with research on the specification and verification of concurrent computations, and research on programming constructs for information systems. I believe this volume is the first to present new technical work on temporal logic from philosophy, software engineering and artificial intelligence; the inclusion of material from artificial intelligence is particularly useful. A coherent account of these fields is given in Dr Galton’s excellent Introduction.

Temporal logic is one of several theories (not all of them mathematical theories) that have arisen in philosophy and are finding application in computer science. I think the case of temporal logic exemplifies rather well how progress in an academic field can depend on the importation and adaption of ideas from an apparently unrelated field. We are reminded that intellectual life is not to be regimented; also that speculative research is a precious possession of our universities. Let it not go unnoticed by the readers of this book that for the past few years, while computer science has received every encouragement, philosophy has been discouraged and dismantled in many universities. I think that, in less than a decade, this disregard for philosophy will be seen as unforgivable stupidity.

The book is based on a conference organised by Dr Galton on behalf of the Centre for Theoretical Computer Science at the University of Leeds. The aim of the Centre, like that of the meeting and the book, is to help create an interdisciplinary milieu in which to experiment with research relevant to the foundations of computer science. I hope the reader will enjoy the different contributions and will be inspired by the variety of problems they raise.

J. V. Tucker
Director of the Centre for Theoretical Computer Science
University of Leeds

PREFACE

This book has arisen from a conference on Temporal Logic and its Applications held at the University of Leeds in January 1986 under the auspices of the then newly created Centre for Theoretical Computer Science. Some sixty delegates attended the conference, drawn mainly from computer science, philosophy, and mathematics. Of the papers in this book, those by Barringer, Galton, and Hale are directly based on presentations given at the conference, the other papers being also closely related to material presented there.

As the reader of this book will discover, temporal logic is a field which, having originated within philosophy, has now proved to be of relevance to several distinct areas in computer science. This is, I believe, the first publication in which all of these aspects of temporal logic are treated together. It is to be hoped that the book will provide a stimulus to further inter-disciplinary collaboration, not only as regards temporal logic itself but also in connection with other logical and philosophical issues which lie at the interface between computing and philosophy.

Each of the chapters in the book is entirely self-contained, and can be read independently of all the others. It is recommended, however, that the reader who is unfamiliar with temporal logic in any form should first read the introductory chapter.

Antony Galton
## CONTENTS

1 TEMPORAL LOGIC AND COMPUTER SCIENCE: AN OVERVIEW
   Antony Galton

<table>
<thead>
<tr>
<th>Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 The philosophical background</td>
<td>2</td>
</tr>
<tr>
<td>1.2 The modal approach</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Tense Logic and the topology of time</td>
<td>5</td>
</tr>
<tr>
<td>1.4 The two approaches—rivals or allies?</td>
<td>6</td>
</tr>
<tr>
<td>1.4.1 The two approaches as rivals</td>
<td>6</td>
</tr>
<tr>
<td>1.4.2 The two approaches as allies</td>
<td>8</td>
</tr>
<tr>
<td>1.5 Some related work on the logic of temporal discourse</td>
<td>11</td>
</tr>
<tr>
<td>1.5.1 Aspect</td>
<td>12</td>
</tr>
<tr>
<td>1.5.2 Action</td>
<td>14</td>
</tr>
<tr>
<td>1.6 Reichenbach's theory of tenses</td>
<td>17</td>
</tr>
</tbody>
</table>

2 Temporal logic in artificial intelligence and cognitive science    18
| 2.1 Bruce's 'Chronos'                                        | 18 |
| 2.2 Kahn and Gorry's 'Time Specialist'                      | 19 |
| 2.3 McDermott's temporal logic                              | 20 |
| 2.4 Allen's theory of time                                  | 23 |
| 2.5 Other recent approaches                                 | 26 |

3 The use of temporal logic in software engineering          27
| 3.1 Program verification                                    | 28 |
| 3.1.1 Floyd's inductive assertion method                    | 28 |
| 3.1.2 Hoare's axiomatization of the inductive assertion method | 30 |
| 3.1.3 A new approach: Burstall's intermittent assertion method | 34 |
| 3.2 Further developments of the modal approach              | 35 |
| 3.3 Syntactic and semantic developments                     | 37 |
| 3.3.1 Linear and branching time                              | 37 |
| 3.3.2 Fairness and related properties                       | 39 |
| 3.3.3 Extended temporal logic                               | 40 |
| 3.3.4 Interval semantics                                    | 41 |

3.4 Pragmatic developments                                  42
| 3.4.1 Automatic verification                               | 44 |
| 3.4.2 Automatic synthesis                                  | 44 |
| 3.4.3 Compositional specification                          | 45 |
| 3.4.4 Temporal logic as a programming language             | 46 |

4 Concluding remarks                                          48

Acknowledgements                                               50

References                                                     50
## CONTENTS

### 2 THE USE OF TEMPORAL LOGIC IN THE COMPOSITIONAL SPECIFICATION OF CONCURRENT SYSTEMS

**Howard Barringer**

1 Introduction ........................................ 53
1.1 The formal paradigm ................................ 54
1.2 Specifications ...................................... 56
1.3 Hierarchic development and compositional specification .......... 57
1.4 Organization ....................................... 59
2 Compositional temporal specifications .......................... 60
2.1 An action language .................................. 61
2.2 A temporal logic with fixed points ...................... 62
   2.2.1 Syntax ......................................... 62
   2.2.2 Models and interpretations ....................... 62
   2.2.3 Axioms and proof rules .......................... 65
   2.2.4 Comments ....................................... 67
2.3 Action language semantics ................................ 67
   2.3.1 General approach ............................... 67
   2.3.2 Action language temporal semantics .......... 68
   2.3.3 Example ....................................... 71
2.4 Action language proof system .......................... 73
2.5 Introducing hiding ................................... 75
3 Further developments ................................... 76
3.1 Distributed variables ................................ 77
3.2 Fully-shared variables ................................ 79
3.3 Synchronized actions ................................ 80
3.4 Channel-based communication ........................... 80
4 Some trouble with abstraction ............................. 81
4.1 The problem ........................................ 81
4.2 Example ............................................ 82
4.3 Some formal notions .................................. 83
4.4 Possible approaches to abstractness ...................... 85
   4.4.1 Collapse of the expanding state ............... 86
   4.4.2 Next-change logic—EL .......................... 87
   4.4.3 Dense time ..................................... 88
Acknowledgements ........................................ 89
References ............................................... 89

### 3 TEMPORAL LOGIC PROGRAMMING

**Roger Hale**

1 Introduction ........................................ 91
2 Interval temporal logic ................................ 92

### 4 THREE RECENT APPROACHES TO TEMPORAL REASONING

**Fariba Sadri**

1 Introduction ........................................ 121
2 Kowalski and Sergot's calculus of events ..................... 122
   2.1 An example ....................................... 122
   2.2 Partial event descriptions ....................... 129
   2.3 Notation ......................................... 131
   2.4 The general axioms ................................ 133
   2.5 Derivation of end points for time periods .......... 136
   2.6 A special case of the Event Calculus ............... 137
3 Lee et al.'s logic of time and events ....................... 140
   3.1 An example ....................................... 140
   3.2 Notation ......................................... 141
   3.3 The general axioms describing the holding of relationships and the persistence of properties ........... 144
   3.4 Possible problems with axioms (L3) and (L4) .......... 145
   3.5 The relationship between Event Calculus rules and Lee et al.'s axioms ........................................ 147
   3.6 Application of Lee et al.'s logic to the project assignment example ........................................ 149
   3.7 Partial event descriptions .......................... 151
4 James Allen's temporal logic ................................ 153
   4.1 An example ....................................... 153
   4.2 Some characteristics of Allen's logic ............... 156
INTRODUCTION

In recent years it has increasingly been argued that philosophers can no longer afford to ignore the fundamental innovations in concept and method introduced by practitioners of various disciplines in Computer Science, notably in the study of Artificial Intelligence. Work in many of the traditional problems of philosophy, particularly those associated with language and mind, ought, according to this line of argument, to receive fresh impetus from the insights of those who have sought to tackle related problems from a practical, computational viewpoint (see, for example, Sloman, 1978).

Whatever one may think of the specific claims of Aaron Sloman and others who have argued along these lines, one cannot seriously deny that philosophy always has drawn inspiration from close association with other intellectual disciplines, and there is no reason to suppose that computer science should prove an exception in this regard; there are already signs of growing interest in all matters computational within the philosophical community, and there is every expectation that this trend will continue into at least the immediate future. Conversely, it is just as clear that computer science has a good deal to learn from philosophy, and indeed computer scientists are showing ever greater awareness of and interest in those aspects of philosophy with the greatest relevance to their own discipline.

Temporal Logic affords a particularly good illustration of the sort of close association between two disciplines I have alluded to. It arose, in the form of Tense Logic, purely within the domain of philosophy, and was pursued for many years by philosophers and logicians without any regard to its possible applications outside their subject. Yet over the last decade or so it has become apparent that temporal logic has valuable potential as a formal tool for use in
both artificial intelligence and software engineering. This has come about, I think, because computer science as a whole is at once highly formal and deeply rooted in the practicalities of everyday life, so that a formalism designed to handle so pervasive a feature of everyday life as time has a natural role to play in it.

This introductory survey falls into three main sections. In the first, I examine the philosophical background in which temporal logic was first developed; next, I consider the applications of ideas from temporal logic to the fields of Artificial Intelligence and Cognitive Science, in which temporal logic is used to enable computer programs to reason about the world; and lastly, I examine the further development of temporal logic in software engineering as a tool to enable the world to reason about computer programs.

1 THE PHILOSOPHICAL BACKGROUND

1.1 The First-order Approach to Time

In mathematics, time has traditionally been represented as just another variable, indeed time is the independent variable par excellence, the dependence of a variable on time being the paradigmatic example of the functional dependence of one quantity on another. This is because historically one of the most important activities of mathematicians has been that of giving a formal quantitative account of physical processes, and such processes by their very nature always involve time. In physics, times are generally represented by real numbers, but in other applications integers or yet other mathematical objects may be more appropriate. Any such model of time involves the use of individual terms to designate times, and the logical apparatus employed for reasoning with such terms is invariably the first-order predicate calculus. Thus far, then, there is no need to speak of temporal logic; if times are designated by terms in a first-order theory, then this is just ordinary logic applied to a subject matter which happens to include time. And this is adequate for physics and the other mathematical sciences.

Since Frege, however, philosophers have been interested in logic particularly in its relation to language. Here, unlike in applied mathematics, time does not appear in the guise of just another variable. We do, to be sure, have terms for designating particular times—clock times, calendar dates, and so on—but this way of encoding temporal information looks from a linguistic perspective far less fundamental than the stock of temporal connectives ('when', 'while', 'before', 'after', 'since', and 'until') and adverbs ('now', 'then', 'always', 'sometimes', 'never', 'soon', etc.) which form some of the most basic vocabulary of every language. And of course temporal notions are tightly woven into the grammatical fabric of most languages through the tenses of the verb.

Despite this, the first modern attempts to subject the temporal element of language to a logical treatment inherited, apparently uncritically, the mathematical paradigm of first-order terms denoting individual times; and all temporal features of language were regimented into the forms available within this paradigm.

Thus we find Bertrand Russell defining change as 'the difference, in respect of truth or falsehood, between a proposition concerning an entity and a time $T$ and a proposition concerning the same entity and another time $T'$, provided that the two propositions differ only by the fact that $T$ occurs in the one where $T'$ occurs in the other.' (Russell, 1903, §442). It is important to note here that Russell speaks of the truth or falsity of a proposition concerning an entity and a time—in other words of the eternal truth-value of a proposition of the form $f(x, t)$—rather than of the truth or falsity of a proposition concerning an entity at a time—which would be the temporary truth-value, at $t$, of a proposition of the form $f(x)$.

Quine, who thoroughly endorses this Russellian style of analysis, offers such transcriptions as

$$\neg(\exists x)(x \text{ is a time } \land \text{ Jones is ill at } x)$$

for 'Jones is never ill' (Quine, 1965, p. 91), and goes a step further than Russell, replacing the latter's idea that properties should be construed as relations between entities and times by the idea that they should be construed still as properties, but properties of temporal stages of entities, substituting something like $f(x \cdot t)$ for $f(x, t)$. Here $x \cdot t$ is merely a 'slice', at right angles to the time axis, of the extended four-dimensional object $x$.

1.2 The Modal Approach

The methods of analysis proposed by Russell and Quine effectively remove time from the domain of logic, by treating temporal elements of a sentence on a par with other non-temporal elements. The first suggestion that a logical calculus might be constructed specifically in order to handle inferences involving time, thereby according temporal elements logical status, appears to have been made by Findlay in 1941. Findlay's suggestion arose in connection with a critical assault on McTaggart's notorious proof of the unreality of time; but it was not taken up until the early 1950s, when Prior constructed the first tense logic.
Prior had been impressed by a remark in a book review by Geach to the
effect that the ‘p at t’ style of analysis was quite alien to a discussion of ancient
and mediaeval philosophy, which knew nothing of such analyses. Now
expressions like ‘p at t’ had been used by Benson Mates in his discussion of some
tories of the Stoic logician Diodorus Chronus, and Prior hoped to be
able to recast Mates’s discussion in terms not involving the ‘p at t’ style of
analysis which Geach had stigmatized as anachronistic.

Diodorus apparently sought to reduce modal notions to temporal ones,
defining the possible as ‘what is or will be’, the necessary as ‘what is and
always will be’, and the impossible as ‘what is not and never will be’. Prior was
interested to discover which of the existing formal systems of modal logic was
implicit in Diodorus’s definitions, and realized that, in his own words, ‘to get
a logic of the possible from its definition in terms of the future, one must also
have a logic of futurity. The construction of a calculus of tenses could not wait

The seminal paper in which Tense Logic first saw the light of day, Prior
(1955), already contained the now-familiar notations Fp (for ‘It will be
the case that p’), Gp (for ‘It will always be the case that p’), Pp (for ‘It has been
the case that p’), and Hp (for ‘It has always been the case that p’), as well as the
identifications of Gp and Hp with ¬F¬p and ¬P¬p respectively.

In this initial paper, Prior showed that, given a number of intuitively
plausible postulates concerning the logic of futurity, the Diodorean modal
system obtained by defining

\[ \Diamond p \equiv \text{def } p \lor Fp \]

after the fashion of Diodorus is at least as strong as the Lewis system S4, but
weaker than S5. To get a system as strong as S5, it is necessary instead to
extend the Diodorean account of possibility by including the past as well as
the future, i.e.

\[ \Diamond p \equiv \text{def } Pp \lor p \lor Fp. \]

This is not the place to explore the detailed ramifications of this initial
work on the relationship between tense and modality; an excellent account
can be found in Prior (1967), especially Chapter 2. Suffice it to say that
because of the circumstances attending its birth, much of the early work in
Tense Logic was devoted to articulating this relationship in detail. In fact,
Tense Logic is generally regarded as a kind of modal logic, differing from
ordinary modal logics in possessing two sets of operators, one for the past
and one for the future, where ordinary modal logic has only one. And so the
Prior style of analysis of temporal discourse has come to be known as the
modal approach to time, as distinct from the first-order approach typified by
Russell and Quine.

### 1.3 Tense Logic and the Topology of Time

It quickly became apparent to Prior and other workers in tense logic that the
new formalism was potentially a valuable tool for expressing, and exploring
the consequences of, a range of theses about the structure of time, theses
about what in a somewhat loose way of speaking has come to be called the
topology of time. Examples of questions which have been considered are:

(a) Is time discrete or continuous?
This question can be taken at the ontological level as ‘Are there really time
atoms?’ or as a question about suitable representations within a given
domain—for example, if one is dealing with facts which only vary on a time-
scale consisting of whole numbers of days, then one can take a day as an
indivisible unit and use a discrete time model. Again, suppose we have n
property-variables, each of which is present or absent at any one time, and
that we define an epoch to be a maximal period over which none of the
variables changes value. Then the succession of epochs constitutes a discrete
time structure imposed on what may very well be an intrinsically continuous
underlying temporal framework.

Note that if time is treated as discrete, then we can introduce a new
operator \( \text{next} \), read ‘at the next moment’. This operator is widely used in the
computer science applications to be described below.

(b) Is time bounded or unbounded?
Note that this is not quite the same as the question whether time is finite or
infinite in extent. The latter question only has meaning if a metric has been
defined on the time line. But even without a metric, there are still the
possibilities that either every time is succeeded by a later time (unbounded in
the future) or there is a last moment in time (bounded in the future), and
likewise in the direction of the past.

(c) Is time linear, parallel, or branching?
Here we are dealing with the gross topological properties of time. The idea of
branching time has been put forward as a way of handling uncertainty about
the future, the idea being that from each moment forward there exist many
possible alternative futures. Unfortunately, this picture does not succeed very
well in capturing the idea that of all the possible alternative futures there is
one privileged ‘actual’ future. In the branching time systems, one can easily
express the propositions that it may be going to be the case that \( p \) and that it
is bound to be the case that \( p \), but it is not so easy to express the proposition
that it will be the case that \( p \), so a branching time model is only likely to
appeal to someone who does not think this third proposition has any meaning distinct from the other two.

Against this, there seems to be something very compelling about the idea of the actual future, distinct from all other possible futures. To accord actuality to all the branches does not seem to be very meaningful unless there is some way of communicating between the branches so that they can interact—this is the stuff of science fiction, as for example in some of the short stories of John Wyndham—and not something that many philosophers, at least, would be prepared to take seriously.

As for parallel times: we could perhaps regard this as a model of the different subjective time-scales of different people, but again, in order to set up a formal calculus here it would seem necessary to establish correlations between the different times, and the robustness of such correlations is a measure of how well-founded is the idea of an objective time underlying all the different subjective time-lines.

Circular time is an interesting possibility, too, for in this, past, present and future coalesce. It is hard to take circular time seriously as an account of physical time, but the associated logic could be useful in reasoning about repetitive processes, for example the effectively endless repetition of cycles in the traffic signals at a road junction.

All these topological issues are discussed at length by Newton-Smith (1980).

1.4 The Two Approaches—Rivals or Allies?

The modal and first-order approaches may be viewed either as rivals or as allies.

1.4.1 The two approaches as rivals

There have been extensive philosophical debates as to which approach is ‘correct’. Massey went so far as to speak of a Kuhnian paradigm clash (Massey, 1969). Massey invented the now popular terminology of ‘tensers’ and ‘detensers’ to describe the adherents of the modal and first-order approaches respectively. What are the arguments?

(a) Metaphysical arguments about existence and time. On the tenser view there is a crucial distinction between such pairs of sentences as $F(\exists x)f(x)$

and

$(\exists x)f(x)$

(i.e. 'It will be the case that there is something which is $f$' and 'There is something of which it will be the case that it is $f$'). The former sentence might be true if something that does not yet exist is later on going to exist and be $f$, whereas the latter can only be true if there already exists something that is later going to be $f$.

For the detenser, these formulae come out as something like $$(\exists t)[later(t, now) \land (\exists x)f(x, t)]$$

and

$$(\exists x)(\exists t)[later(t, now) \land f(x, t)]$$

and these are easily seen to be equivalent in first-order logic; so for the detenser the alleged distinction cannot be upheld. This means that tensers (or at least those who accept the distinction) and detensers must take a radically different view of the relationship between existence and time. For the detenser, anything which exists exists timelessly; its temporal qualifications are a result of its standing timelessly in relation to certain times. In this view of the world, time is treated as a dimension on a par with the three dimensions of space; it is generally felt that the theory of Relativity lends support to this way of regarding time, although the latter is in no way dependent on the former. For the tenser, this ‘four-dimensionalism’ is, if not simply incoherent, then at least radically deficient in that it leaves totally out of account that transience which, according to the tenser, is an essential feature of time and temporal phenomena.

(b) Conceptual arguments. A different line that is taken by tensers is that if we want our temporal logic to mirror conceptual priorities—which from a philosophical point of view is surely desirable—then we must treat tenses and all the other adverbial paraphernalia of temporal reference as more basic than dates and times. An eloquent exponent of this point of view has been Geach, who argued that it is perverse to try to analyse ‘grass roots temporal discourse’ (i.e. tenses and simple temporal relations like ‘before’ and ‘after’) in terms of the ‘vastly more complex notions’ implicit in the first-order approach (Geach, 1965).

Of course, even if one grants that Geach’s point is a compelling one from the point of view of ultimate philosophical analysis of the structure of our temporal concepts, the detenser may still with some propriety claim that that point of view is not the only one in which there is need to formalize temporal language, and that from the point of view of physics, say, or computer databases, it would be hopelessly cumbersome to analyse all times and dates into Geach’s ‘grass-roots’ style of discourse, even if it could be agreed just what is the correct way of doing this.
(c) **Logical arguments.** There is a persistent undercurrent in much modern logic to the effect that first-order logic is in some way definitive, that nothing is truly intelligible unless cast in first-order form. This view has been championed by Quine and by Davidson, amongst others (in Susan Haack’s words, for Quine extensionality is the touchstone of intelligibility)—the point being that first-order logic is extensional, whereas modal logics are not. First-order treatments have the advantage that first-order logic is known to be complete with respect to its standard model theory (as against, e.g., second-order logic—which although necessity compels us to continue doing arithmetic), and there exist decision procedures and proof procedures for substantial and important subsets of it (even though the theory as a whole provably lacks either). Moreover, for computer science, first-order treatments lend themselves readily to implementation as logic programs, especially if they are expressed entirely in terms of Horn Clauses, in which case the implementation in Prolog is practically automatic.

(d) **Expressive power.** Not everything that can be expressed in the first-order approach is expressible in the modal approach, unless additional operators are introduced. On the other hand, the modal approach has the advantage that its expressions are both simpler and closer in spirit to natural language. This issue is treated in greater detail below (pp. 39–41).

1.4.2 The two approaches as allies

The alliance between the two approaches comes about because the first-order approach can be made to serve as a model theory for the modal approach, i.e. in order to give a formal account of the intended meanings of the operators in the latter approach, one typically makes use of formalisms drawn from first-order logic.

This state of affairs is almost inevitable. If asked, what does it mean to say ‘It will be the case that p’, it is very hard to avoid coming up with something like ‘There is some future time at which p is true’. A tenser might say that this is a mere façon de parler, but it is nonetheless impressive that we are able in everyday language to handle quantification over times in a way quite analogous to the way in which we handle quantification over less contentious entities, and without, apparently, lapsing into gross conceptual confusion. Even Prior, the arch-tenser, has to resort to this sort of talk to explain precisely the meaning of formulae in tense logic.

The style of model theory standardly used for Tense Logic is a straightforward adaptation of the well-known Kripke semantics for modal logics (Kripke, 1963), originally due to Kanger (1957). It is noteworthy that this particular semantical paradigm seems a good deal more natural when it is applied to tense logics than when it is applied to modal logics, since of the entities invoked in each case, times are a part of our everyday gamut of concepts in a way that possible worlds (which play the corresponding role in the modal semantics) just aren’t.

It should be noted that Prior himself had little to do with the model theory of tense logic, largely preferring to confine his technical discussion to the proof theory. Accounts of the model theory can be found in Rescher and Urquhart (1971) and in McArthur (1976). Rescher and Urquhart also give proof systems for both linear and branching tense logic based on the method of semantic tableau. Alongside Prior (1967), Rescher and Urquhart’s book has assumed the status of a classic in the literature on temporal logic; it remains to be seen whether the recent scholarly and comprehensive book by van Benthem (1985) will achieve a similar distinction.

The formal semantics for Tense Logic is based on the following ‘translations’:

\[
Pp \text{ is true at } t \iff (\exists t')[(\text{later}(t, t') \land p(t')]
\]

\[
Fp \text{ is true at } t \iff (\exists t')[(\text{later}(t, t') \land p(t')]
\]

\[
Hp \text{ is true at } t \iff (\forall t')[\text{later}(t, t') \rightarrow p(t')]
\]

\[
Gp \text{ is true at } t \iff (\forall t')[\text{later}(t, t') \rightarrow p(t')]
\]

Using these, we can straightforwardly convert any tense-logical formula into first-order form, e.g. the formula

\[
p \rightarrow Gp
\]

becomes

\[
p(t) \rightarrow (\forall t')[\text{later}(t', t) \rightarrow (\exists t'')[\text{later}(t', t'') \land p(t'')]].
\]

Since the latter is a theorem of first-order logic, the tensed version had better be a theorem of tense logic too, and indeed it, or something equivalent, is included as an axiom in all systems of tense logic.

Other formulae which translate into first-order theorems are \(p \rightarrow Hp\), \(G(p \rightarrow q) \rightarrow (Gp \rightarrow Gq)\), and \(H(p \rightarrow q) \rightarrow (Hp \rightarrow Hq)\). This list is complete in the following sense: if we take them as axiom schemes for a tense-logical system, together with the tautologies of the propositional calculus, and if we posit as rules of inference

- MP: from \(\vdash \alpha\) and \(\vdash \alpha \rightarrow \beta\) infer \(\vdash \beta\)
- RG: from \(\vdash \alpha\) infer \(\vdash Ga\)
- RH: from \(\vdash \alpha\) infer \(\vdash H\alpha\)
then the theorems of the resulting system are precisely those tense-logical
formulae which translate into theorems of first-order logic. The system of
tense logic here outlined is known as Minimal Tense Logic, and forms the
basis for most other systems that have been studied.

Any tense-logical formulae that are not theorems of Minimal Tense Logic
will translate into first-order formulae that are no longer theorems of first-
order logic, for example the tense-logical formula

\[ FFp \rightarrow Fp \]  

which says that whatever will be future is already future, translates into the
first-order formula

\[ (\exists t')[later(t, t') \land (\exists t'')[later(t', t'') \land p(t'')]} \rightarrow
\[ (\exists t'')[later(t, t'') \land p(t'')]} \]  

which is not a first-order theorem.

If (1) is to be added as an axiom schema, thus extending Minimal Tense
Logic, the consequence in the first-order translation is that (2) must be
stipulated to hold for all times \( t \) and for all predicates \( p \) (since if (1) is to be
treated as an axiom schema, \( p \) must be regarded as a schematic letter doing
work for any formula that might be substituted for it), so that in effect the
translation of (1) when regarded as an axiom schema is now the second-order
formula

\[ (\forall p)(\forall t)([\exists t'][later(t, t') \land (\exists t'')[later(t', t'') \land p(t'')] \rightarrow
\[ (\exists t'')[later(t, t'') \land p(t'')] \]  

which again fails to be a theorem of second-order logic.

In general, then, the addition of further axioms to Minimal Tense Logic
amounts to the assertion of substantial second-order theses about the earlier-
later relation. In the example cited, it turns out that this second-order thesis is
in fact equivalent to a first-order one, namely

\[ (\forall t)(\forall t')(\forall t'')([later(t, t') \land later(t', t'')] \rightarrow later(t, t'')] \]  

which asserts the transitivity of the relation ‘later’. For the most part, it is
first-order statements like (4) rather than second-order ones like (3) that are
of interest to us, for they correspond to intuitively ‘salient’ properties that a
temporal ordering may possess, such as transitivity, density, boundedness,
and so on. It is therefore of interest to know which tense-logical formulae,
when taken as axiom schemas, yield first-order properties.

Not all do; a simple example which does not is McKinsey’s formula
\( GFp \rightarrow FGp \). Van Benthem (1983, p. 158) has given an exact characterization
of those tense-logical formulae which, when taken as axioms, give rise to first-
order formulae. This tells us which formulae are first-order definable; but
there still does not exist a general effective procedure for obtaining these first-
order equivalents when they exist.

The converse task, that of finding a tense-logical formula to correspond to
a given first-order formula, is also of interest, because it bears rather strongly
on the issue of the relative expressive powers of the two methods of
representation. In fact it is not hard to find first-order properties of temporal
orderings which are not tense- logically definable, a well-known example being
the formula

\[ (\forall t)\neg later(t, t) \]  

which expresses the irreflexivity of the temporal ordering. This means, in
effect, that no tense-logical system can constrain its models to be irreflexive.

Once again, it can be shown that there is a precisely defined class of
transformations such that a first-order sentence making use only of the
predicates ‘later’ and ‘=’ is tense- logically definable if and only if it is
preserved under that class (see van Benthem 1983, p. 161).

1.5 Some Related Work on the Logic of Temporal Discourse

The fact that there are first-order formulae in ‘later’ and ‘=’ which are not
tense- logically expressible means that the language of tense logic is expres-
sively incomplete with respect to the first-order temporal logic. This naturally
prompts the question whether the tense-language can be extended so as to
become expressively complete in this sense. An affirmative answer is supplied
by the work of Kamp (1968), who developed a temporal logic using the
binary operators \( S \) and \( U \), read ‘since’ and ‘until’, whose translations into
first-order language are

\[ Spq \text{ is true at } t \iff (\exists t'')[later(t, t') \land p(t'')] \rightarrow q(t'')] \]  

\[ Upq \text{ is true at } t \iff (\exists t')[later(t', t) \land p(t') \land
\[ (\forall t'')[later(t', t') \land later(t, t'') \rightarrow q(t'')] \]. \]

Informally, \( Spq \) is true now so long as \( p \) has been true at some past time and \( q \)
have been true ever since then; while \( Upq \) is true now so long as \( p \) will be true at
some future time, and \( q \) will be true up to then. Kamp showed that, assuming
continuity and strict linearity of the temporal order, any first-order temporal
formula can be expressed as a formula of the \( S, U \)-calculus.

Note that it would be misleading simply to equate Kamp’s operators ‘\( S \)’
and ‘\( U \)’ with the English conjunctions ‘since’ and ‘until’. There are at least
two major reservations one might have about such an equation. Consider an English sentence such as ‘I will be unhappy until you come’, and the putative rendering into Kamp’s notation as $Upq$, where $q$ is ‘I am unhappy’ and $p$ is ‘You come’. The first problem is that whereas $Upq$ implies $Fp$, the English sentence does not imply ‘You will come’; the second is that the English sentence suggests that once you have come, I shall no longer be unhappy, whereas this is obviously not an implication of $Upq$. Nonetheless, it does seem that $S$ and $U$ come as near to capturing the essence of ‘since’ and ‘until’ as any relatively simple logically defined pair of connectives could hope to do.

Other temporal connectives have also been studied by philosophers, with varying degrees of formality. Two classic treatments are Anscombe’s largely informal study of ‘before’ and ‘after’, and von Wright’s formal $T$-calculi, embodying a connective $T$, to be read as ‘and then’ if time is treated as continuous, and as ‘and next’ if time is discrete. Both these studies are of interest because they link up with other important areas in the study of temporal logic. Anscombe’s with the logic of aspect, and von Wright’s with the logic of action; and both these areas reach beyond the domain of tense logic proper because of the prominence assumed in them by the notion of an event.

1.5.1 Aspect

Anscombe (1964) noted that, contrary to what one might imagine, ‘before’ and ‘after’ are not strict converses of each other. For example, it is true that

$$Haydn \text{ was alive before } Mozart \text{ was alive}$$

and that

$$Haydn \text{ was alive after } Mozart \text{ died}$$

but it is not true either that

$$Mozart \text{ was alive after } Haydn \text{ was alive}$$

or that

$$Mozart \text{ died before } Haydn \text{ was alive}$$

whereas if ‘before’ and ‘after’ were strict converses, one would expect (5) to be equivalent to (7), and (6) to (8).

On the other hand, the sentences

$$Haydn \text{ was born before } Mozart \text{ was born}$$

and

$$Mozart \text{ was born after } Haydn \text{ was born}$$
do appear to be pretty nearly equivalent, suggesting that here ‘before’ and ‘after’ are converses. Anscombe noted that ‘before’ and ‘after’ are only true converses when the clauses they link report the occurrence of instantaneous events, and it is this observation more than any that links her work with the study of aspect.

Primarily, the term aspect refers to those elements of grammatical structure which draw attention to how a state of affairs or event is presented in a sentence, so that for example the perfective aspect (represented in English by the simple past tense as in ‘I wrote a letter’) presents an event as a completed whole, whereas the imperfective aspect (‘I was writing a letter’) presents it rather as something in progress, with no implication as to whether or not it eventually comes to completion. This simple description belies the complexity of what in fact has proved to be an enormously difficult and challenging area, which has increasingly during recent years attracted the attention of linguists, philosophers, and logicians.

For our present purposes, the most important idea is that the existence of aspeclual distinctions gives rise to a taxonomy of verb-types according to their so-called inherent aspect or aspeclual character (also known by the German term Aktionsart). Some verbs, or verb-phrases, refer to states (as in ‘The cake is in the oven’), some to processes or activities (as in ‘Mary is baking’), some to events which take time (as in ‘Mary baked a cake’), and some to instantaneous events (as in ‘The light went out’). This four-fold division was given by Vendler (1967), who labelled the four types ‘states’, ‘activities’, ‘accomplishments’, and ‘achievements’ respectively. Vendler was not the first to attempt a classification along these lines, and indeed, the beginnings of such a classification are already to be found in Aristotle (cf. Ackrill, 1965); but Vendler’s classification seems to have been the most influential, and the variant schemes that have been worked out from time to time since then have seldom departed far from Vendler’s own.

Much of the work on aspeclual character has been, like Vendler’s, discursive in nature, with little attempt to integrate the results into a formal system of temporal logic. Most of those who have attempted to do this have ended up rejecting the standard semantics for tense logic, which is based on instants, in favour of a semantics based on intervals. Interval semantics has been taken up by Cresswell (1977), Dowty (1979), Humberstone (1979), and Richards (1982).

The idea motivating the shift from instants to intervals appears to be that since some events typically take time, a sentence reporting an event ought to be assigned a truth-value, not relative to an instant (which is as it were too small to accommodate the whole event), but relative to an interval—the sentence being true on an interval just so long as the event it refers to takes up exactly that interval and no more.
I argued in Galton (1984) that the 'marriage of propositions with intervals is at best an unhappy one' (p. 21); it seemed to me that interval semantics arose from a confusion between something's being true at a time, and its being true of a time. It might well be true of a particular hour-long stretch, say, that I spent writing a letter, but that does not mean that we should adopt a semantical scheme whereby the sentence 'I write a letter' is assigned the value 'true' relative to that hour-long stretch, for no such assignments can shed any light on what it is for a sentence about my letter-writing to be true at the time that it is uttered. Essentially the same point has been well argued, independently of my own work, by Pavel Tichý (in Tichý, 1985). For further discussion on this, see below (p. 171).

As an account of natural language semantics, then, interval semantics seems to me to be deficient. Despite this, there is, as we shall see, a use for an interval-based semantics of temporal logic within computer science (see below).

1.5.2 Action

Von Wright's T-calculi (von Wright, 1965, 1966) arose from an attempt to supply a formal basis for philosophical reasonings about action, this being what is required for a rational understanding of the notions of free will and responsibility and the ethical issues with which they are entangled. Much has been written, and continues to be written, in the attempt to elaborate a philosophically sound theory of human action, the key problem being perhaps one of definition: what is it about one set of bodily movements that makes it an action, while another apparently identical set is not?

Von Wright's work in this area focussed attention on the relationship between action and change. Some actions consist in bringing about a change, others in preventing a change from occurring; but in either case the notion of change is crucial to an understanding of action, so that, according to von Wright, it will not be possible to develop a logic of action without first constructing a logic of change; and this is what his T-calculi were designed to provide.

Von Wright's starting point was the Russellian idea that change can be characterized in terms of the difference in truth-value of the same proposition at different times (or, as Russell himself put it, the difference in truth-value of propositions about different times but otherwise identical). A change in respect of a proposition p thus comes about either through p's being true and then false, or the other way round. It was to express this situation that von Wright introduced the binary connective T such that pTq is true now if and only if p is true now and q is going to be true at some later time. Of course, for many possible choices of p and q, no change is implied by the truth of pTq at any time; but if they are incompatible with one another, then some change must occur in going from a state of the world in which p is true to a state in which q is true; so that in particular the proposition (¬p)Tq expresses the change involved in p's coming to be true.

A curious feature of this set-up is that it is only possible to express future changes in this way; this is a consequence of propositions of the form pTq being evaluated for truth or falsity at the earlier of the two times involved in the change. The interval semanticists (notably Humberstone) pointed out that the circumstance of p's being true followed by q's being true is something that obtains not at a single moment of time but over an interval: this was one of Humberstone's arguments for basing the semantics on intervals in the first place.

In Galton (1984), I adopted a different approach, focussing attention not on the sequence '¬p then p' (which as Humberstone rightly pointed out requires an interval for its realization) but on the event of transition from ¬p to p, which is by contrast instantaneous. And I noticed that a past occurrence of this event can be expressed by the tense-logical formula

\[ P^*(¬p \land p) \]

whereas a future occurrence of the same event is expressible by

\[ F^*(¬p \land Fp) \]

where P*α and F*α are defined to be equivalent to α ∨ Pa and α ∨ Fa respectively. It was the lack of any common formal element between these two formulae that led me to postulate the necessity of a new kind of temporal logic, Event Logic, in which events can receive a uniform characterisation. Event Logic is the subject of my paper 'The Logic of Occurrence' (this volume, Ch. 5).

Von Wright's T-calculi have been quite influential in providing a starting-point for the development of later approaches to the logic of change, although it seems that the T-calculi themselves are now of little more than historical interest. It should be noted, incidentally, that von Wright approached temporal logic as a tense: his T-formulae contain no temporal references, and thus may have different truth-values at different times.

A stark contrast to von Wright's work is the detenser account of action given by Davidson (1967). This arose out of a concern for logical form. Davidson's fundamental aim was to provide a Tarski-style semantic analysis of natural language. In order to do this, he followed Quine in adopting a thoroughgoing program of reducing all natural language constructions to first-order logical formulae.

An apparently insurmountable obstacle to this program was the problem
of adverbials: what, asked Davidson, is the first-order logical form of a sentence like

\[ \text{Jones buttered toast with a knife} \]  

(9)

It would not do to postulate a three-place predicate \( b(x, y, z) \) to render 'x buttered y with z', for as Anthony Kenny had pointed out (Kenny, 1963, Ch. 8), this would obscure the logical connections of (9) with such an obviously related sentence as

\[ \text{Jones buttered toast at midnight} \]  

(10)

in which 'with a knife' is replaced by a new expression bearing a totally different semantic relation to the whole.

One possibility which has found favour with some researchers is to treat the adverbial phrases 'with a knife' and 'at midnight' as predicate modifiers, that is as functions which convert the predicate 'buttered' into new predicates 'buttered with a knife' and 'buttered at midnight' respectively. Davidson rejected this approach on the ground that it would mean going beyond the confines of a first-order logical analysis.

The ingenious solution proposed by Davidson has won a wide following. Davidson postulated that (9) and (10) assert the existence of certain events: both assert the existence of an event of Jones's buttering toast, but (9) asserts that such an event came about through the use of a knife, whereas (10) merely asserts that it took place at midnight. Thus Davidsonian logical analyses of (9) and (10) look like

\[ \exists x [ \text{butter}(\text{Jones, toast, } x) \land \text{with}(x, \text{knife})] \]

and

\[ \exists x [ \text{butter}(\text{Jones, toast, } x) \land \text{at}(x, \text{midnight})] \].

With these analyses, the logical relationship between (9) and (10) (namely, that they both imply 'Jones buttered toast') becomes an elementary exercise in first-order logic.

Note how thoroughly Davidson here embraces Quine's belief that the logical content of a sentence can only be brought out by casting the sentence in first-order form. In order to do this he has to have terms referring not just to people and things ('Jones,' 'toast,' 'knife') but also to times ('midnight') and, most significantly, to events. It is beside the point that events never get to acquire names, appearing only as the bound variables of quantifiers, since in a fuller analysis, according to Quine, the same would be true for all kinds of entities, their names being paraphrased away by means of Russell's theory of descriptions.

Analyses in Davidson's style have been taken further, and integrated with some of the insights into aspectual character mentioned in the last section, by Barry Taylor, whose recent book 'Modes of Occurrence' (Taylor, 1985) provides a full account of his work on the logic of adverbials and related issues to date.

1.6 Reichenbach's Theory of Tenses

Mention must here be made of the work of H. Reichenbach, whose theory of tense (Reichenbach, 1947) has formed the basis of much subsequent work, including some of immediate relevance to this chapter.

To see clearly the gist of Reichenbach's way of looking at tenses, reflect first that a Prior tense such as \( P \) involves implicit reference to \textit{two times}, namely the time at which a sentence \( Pp \) is to be assigned a truth-value, and a prior time at which \( p \) is being said to have been true. By iterating tense-operators, the number of implicit temporal references can be increased without limit, so that, for example, the truth of a sentence of the form \( PFPp \) presupposes a time \( t_1 \) at which it is true, a time \( t_2 \), earlier than \( t_1 \), at which \( FPPp \) is true, a time \( t_3 \), later than \( t_2 \), at which \( PPp \) is true, and finally a time \( t_4 \), earlier than \( t_3 \), at which \( p \) is true.

Now the essence of Reichenbach's theory is that all the tenses of natural language can be accounted for in a scheme which invokes just \textit{three times} for each tense. Reichenbach calls these times \( U \) (for \textit{utterance time}), \( R \) (for \textit{reference time}), and \( E \) (for \textit{event time}). On this scheme, the future perfect tense seen in 'I shall have gone' can be analysed as presenting us with a set-up in which \( U \) (the time at which the sentence is uttered) precedes \( R \) (the time spoken of, at which it is to be true that I have gone), and \( R \) follows \( E \) (the time at which I actually go), the former relation being signalled by the future element 'shall', the latter by the perfect element 'have + past participle'.

A notable success of Reichenbach's method was its analysis of the distinction, in English, between the simple past ('I went') and the perfect ('I have gone'). In both, the event time precedes the utterance time, but whereas the reference time coincides with the event time in the simple past, it coincides with the utterance time in the perfect. Assuming that temporal adverbials attach to the reference time of a sentence, this neatly explains why, for example, we can say

\[ \text{I did it yesterday} \]

(11)

\[ \text{I did it today} \]

(12)

\[ \text{I have done it today} \]

(13)

\[ \text{I have done it yesterday,} \]

(14)

but not
since while a past reference time may fall within either yesterday or today, thus explaining the acceptability of (11) and (12), a present reference time can only fall within today, explaining the acceptability of (13) but not (14).

Reichenbach’s work occupies a somewhat ambiguous position in this admittedly interdisciplinary research area. On the one hand, the striking simplicity of the schematicism and its genesis in a work on symbolic logic suggest that the theory ought to be regarded as a part of logic, and thus within the domain of philosophy; on the other hand, the enterprise does seem to embody a substantive thesis about language, in that it explicitly denies that logically possible tense constructs such as the tense logical formula \( PFp \) cited earlier will find expression in the grammatical structure of natural language — and indeed it does seem to have been amongst linguists rather than amongst logicians that Reichenbach’s theory has proved most influential.

2 TEMPORAL LOGIC IN ARTIFICIAL INTELLIGENCE AND COGNITIVE SCIENCE

One of the areas in which Computer Science has found a use for temporal logic is Artificial Intelligence (AI). Whether the aim here is to duplicate human intelligence or merely to simulate certain of its effects, the important thing is that human intelligence manifests itself both in what we do and in what we say, so that much AI research has been directed towards the tasks of simulating action and simulating language. For both of these tasks, a proper treatment of time is desirable — in the former case because actions take place in time and require an appreciation of the logical structure of temporal facts for their proper planning and execution, and in the latter because, as noted earlier, so much of natural language is pervaded, both at the lexical and at the grammatical level, by temporal notions.

2.1 Bruce’s ‘Chronos’

An early attempt at mechanizing part of our understanding of time within an AI context was that of Bruce (1972), who presented a formal model of temporal reference in natural language. Built upon a first-order logical base, the model is essentially a highly general theory of tense, in which Reichenbach’s idea of a tense as a relation holding between three points in time is generalized to relations of arbitrary complexity on intervals, the relations being built up out of a basic set of seven binary relations by logical conjunction. In context, the intervals referenced by a given tense may be understood as doing duty for events, an event here being understood as an individual occurrence occupying just the interval associated with it.

Temporal expressions in natural language are mapped onto this abstract theory by means of a rather crude translation procedure whereby each element of temporal discourse is taken as the mark of a single formal relation in the model, complex natural language expressions thus corresponding to complex formal tenses by compounding the translations of the components appropriately. Thus, to formalize a sentence like

John was to have gone

we note that the expression ‘was to’ corresponds in Bruce’s theory to a formal relation which may be briefly represented as ‘after before’, while the past participle (in ‘gone’) corresponds to ‘after’, so that the ‘tense’ of our example is ‘after before after’, which in Bruce’s first-order notation comes out as

\[ \text{after}(s_1, s_2) \land \text{before}(s_2, s_3) \land \text{after}(s_3, s_4). \]

Here \( s_1 \) and \( s_4 \) correspond to Reichenbach’s utterance time and event time respectively, while \( s_2 \) and \( s_3 \) are both reference times, there being no limit, in principle, to the number of distinct reference times possible in Bruce’s theory.

Bruce used his model as the basis for a question-answering program called ‘Chronos’, which seems to have performed satisfactorily within the limitations of its method. These limitations derive in part from the over-simple nature of the translation procedure, which does not take account of the facts that, first, there is no easy way of telling whether a given natural language expression is functioning in a particular context as a tense-marker or as the bearer of a non-temporal meaning (so that, for example, in some contexts ‘He was to go’ means ‘It was the case that he was later going to go’ — a temporal meaning of the tense-logical form \( PFp \) — but in other contexts the same clause may mean ‘He was under an obligation to go’ or ‘There was an arrangement or plan that he should go’), and, second, related to this point, a given tense-marker may indicate different tenses in different contexts (so that, for instance, Bruce’s ‘Chronos’ would presumably be unable to handle a sentence like:

If the employment situation hasn’t improved by the time the prime minister calls the next general election, there will be a change of government

in which the present perfect and simple present tenses of the subordinate clauses have temporal references of the form ‘before after’ and ‘before’ rather than the ‘after’ and ‘equal’ which Bruce’s translation procedure would, if I understand it aright, assign to them).

2.2 Kahn and Gorry’s ‘Time Specialist’

After 1972, not much appears to have been published on modelling temporal inference in AI until the appearance of a paper entitled ‘Mechanizing
Temporal Knowledge’ by Kahn and Gorry (1977). In this paper, Kahn and Gorry introduce the idea of a time specialist, an autonomous body of problem-solving routines specifically designed to handle temporal matters; and they envisage that, once constructed, the time specialist could ‘be placed in the service of a larger problem-solving program to deal with the temporal questions that arise in the domain dealt with by the latter’ (Kahn and Gorry 1977, p. 88).

Unlike Bruce, Kahn and Gorry made no attempt to incorporate any form of natural language understanding in their system, the temporal facts to be processed being first translated manually into intuitively rather opaque but formally more manageable LISP-like expressions. The authors acknowledge that our everyday temporal discourse contains a variety of different types of expression, which only with a certain artificiality can all be regimented into a uniform style of analysis. To accommodate this heterogeneity, their time specialist is endowed with the capacity to organize temporal facts in its memory in more than one way. To be specific, they may be organized (i) by dates, (ii) by referring all temporal phenomena to a basic collection of ‘special reference events’, or (iii) by setting up chains of events ordered by the before-after relation. For a given set of temporal facts one or other of these methods may be the most appropriate; a weakness of the system is that the decision as to which scheme should be used is not automated, but has in every case to be made by the user.

It is debatable how far any of this can be said to involve a temporal logic. Indeed, in their reluctance to relate the natural diversity of temporal expressions to a common underlying formalism, Kahn and Gorry would appear to be eschewing the idea of a temporal logic. None the less, the very concept of a time specialist, with its implication of a determination to abstract the temporal element of reasoning and study it in isolation, has a clear affinity with the ideas motivating the study of temporal logic, and it is perhaps for this reason, more than anything else, that Kahn and Gorry’s work deserves to be mentioned here.

2.3 McDermott’s Temporal Logic

Continuing our survey, we next come to Drew McDermott’s paper ‘A Temporal Logic for Reasoning about Processes and Plans’ (McDermott, 1982). This paper offers a ‘naive theory of time’, in much the same spirit as Patrick Hayes’s ‘naive physics’ (Hayes, 1978). McDermott aims to provide a robust temporal logic to serve as a framework for programs that must deal with time. Clearly the underlying motivation for this is similar to what led Kahn and Gorry to posit the idea of a time specialist as an appropriate goal for AI research; but with McDermott the logical character of the enterprise is made explicit.

Like Bruce, McDermott based his temporal logic on first-order logic; and he makes much of two ‘key ideas’ which dominate his temporal thinking. One of the key ideas is that in order to model continuous change, it is necessary for the time line to be continuous, so that between any two instants there is a continuum of instants. This is achieved by modelling the time line by the set of real numbers. The other key idea is that the future is indeterminate, and hence can only be modelled by having many possible futures for any point in time. To secure this, McDermott does not endow time itself with a branching structure; rather, what he does is to define what happens, or may happen, in time by means of a partially ordered set of states, related to the linear time line by means of an order-preserving date-function from states to instants. A state is, in fact, ‘an instantaneous snapshot of the universe’. The indeterminacy of the future now appears as a condition on the partial ordering of states, namely that if two distinct states both precede some common third state, then one of the two must precede the other. This ensures that the partial order only branches into the future, giving the total set of states a tree-like structure. McDermott names the maximal linear paths through this structure ‘chronicles’; so a chronicle is one possible total world-history.

Certain sets of states are singled out as ‘facts’. The idea here is that a fact is something which may be true in some states of the world and false in others; and the set of states in which it is true is taken as characterizing the fact. This way of defining facts commits us to the view that there cannot be distinct facts true in exactly the same states of the world. If we were restricted to a single chronicle, this limitation would be disastrous, but it might be argued that the branching structure of the totality of states provides just enough ‘leeway’ for the mere pattern of incidence in state-space to individuate facts to the right degree of precision, in much the same way that the ‘possible worlds’ semanticists seek to show that notions which appear intensional when the terms expressing them are interpreted on a universe corresponding to just the actual world become extensional when the interpretation is spread out across the more spacious canvas of a universe corresponding to the totality of all possible worlds.

A set of states can only count as a fact if it obeys certain formal conditions, in particular McDermott’s ‘Axiom 9’, which in effect rules out the possibility of a fact’s changing its truth-value infinitely often over a finite stretch of time. This outlawed situation was described by Prior as ‘a fuzz’, and by Hamblin (1971) as ‘indefinitely fine intermingling’; it is of particular interest to us here because it illustrates how careful one must be, when seeking to axiomatize a particular model of time, to rule out explicitly any undesirable ‘pathological’ consequences of what might seem intuitively the right set of postulates for the
model one has in mind. It is significant, perhaps, that both Hamblin and McDermott seem to have needed external prompting in this matter, for the axiom Hamblin uses to rule out indefinitely fine intermingling was suggested to him by Prior, while McDermott attributes his axiom 9 to Ernie Davis.

In addition of facts, McDermott also discusses events, and notes that they are harder to handle than facts. He goes on to reject two ideas about how events should be handled before settling on his own approach. The first idea he rejects is that an event can be identified as a fact change, i.e. in terms of the facts which characteristically precede and follow the event; and he correctly rejects this idea on the ground that many events just aren't fact changes, an example being Davidson's 'John ran round the track three times'. The second rejected idea is that an event can be identified with the fact that it is taking place; and McDermott notes that this only works for events which 'consist of some aimless thing happening for a while' (what I call 'atelic events' in Galton 1984, pp. 66-68). Other events (and these in fact constitute the great majority of those that are ever of much interest to us) are inhomogeneous in the sense that there is no one fact such that the event just consists of that fact's obtaining for a while.

In the face of these formidable difficulties, McDermott abandons the attempt to give an internal characterization of events in terms of configurations of facts in favour of a wholly external characterization in terms of their temporal incidence, identifying an event as the set of intervals on which it happens. Now this move is not without value (indeed, I myself adopt a very similar ploy in 'The Logic of Occurrence', see Ch. 5, this volume) but it is important to be clear about precisely what its value is. It seems to me quite legitimate to set up a correspondence between events and the sets of intervals on which they occur, in order to reason about the logic of event occurrence by investigating the formal properties of those sets of intervals; I would question the propriety, however, of identifying an event with a set of intervals, as McDermott apparently wants to do.

It is not that such an identification would in principle rule out the possibility of retrieving an internal characterization of the event, for in fact it need not, so long as the branching chronicle-tree is expansive enough to cover every eventuality. If, for example, the event of John's driving from London to Leeds is initially defined as the set of intervals on which this

occurs (in every possible chronicle, not just the 'actual' chronicle), then the internal characterization of this event could be derived as the uniquely most inclusive conjunction of facts true of all the intervals in the set. This would include such facts as that John is in London for a while at the beginning of the event, in Leeds at the end of the event, is driving a car most of the time during the event, and so on. Any 'accidental' features such that, say, the car was blue on every occasion that John actually drove from London to Leeds, will be ruled out by the inclusion of non-actual, but possible, chronicles in which the car is a different colour.

Rather, it is the conceptual ordering that seems to me crucial; an event is what it is because of its internal characteristics. The external characteristics may suffice for certain purposes, but can never amount to a definition, and indeed, the complete set of chronicles required to retrieve the internal characterization from the external one could not, in principle, be obtained unless we already had at our disposal the very internal characterization we are seeking to retrieve.

McDermott glibly avoids this issue, with the result that it is not clear just what is supposed to be achieved by the rejection of two internal characterization and their replacement by the external one; to treat them as comparable in this way is to ignore the fundamental difference in type of the two kinds of characterization, and would seem to betoken a gross confusion as to just what is required of a robust foundation for a temporal—or indeed any other—logic.

2.4 Allen's Theory of Time

Another major figure in the AI approach to temporal logic is J. F. Allen, who in a series of papers appearing over a period of several years has elaborated what he calls a theory of time intended to play the part of a foundation for the sorts of temporal reasoning required in AI type applications. Allen's approach differs from McDermott's in a number of important ways, and it is these differences that I shall particularly select for discussion here. A general introduction to Allen's theory, in the context of a detailed comparison with two other recent theories, can be found in Fariba Sadri's contribution to this volume (Ch. 4).

Allen endorses the Kahn and Gorry idea of a time specialist, but differs from them in not being concerned with dates (Allen, 1981, p. 222). Rather, he is concerned with temporal relationships. Moreover, he rejects instants as the basic unit of temporal reference, preferring to replace them with intervals. His reason for this is that 'the only times we can identify are times of occurrences

2. More precisely, there is no such fact that can be described independently of its being constitutive of the event, for of course if we take an event such as the one reported in the sentence 'John ran round the track three times' then there is a fact, namely the one reported in the sentence 'John is in the process of running round the track three times, and will finish doing so', such that our event does simply consist of that fact's obtaining for a while, but this fact cannot be described independently of the event.

3. A similar criticism has been made by Turner (1984).

and properties (Allen, 1984, pp. 127–128); and such times, according to Allen, are always decomposable into subtimes, and thus must be treated as intervals rather than as instants. Allen thus implicitly rules out the notion of an instantaneous event, and this ruling seems to me an important weakness of the interval-based approach. For further discussion of this point, see Galton (1986).

Bruce's set of seven basic binary relations on intervals is expanded to nine in Allen (1981) and to 13 in Allen and Koømen (1983). These relations are themselves interrelated by a set of transitivity relations, an example of which is that if \( i \) is during \( j \) and \( j \) precedes \( k \), then \( i \) precedes \( k \). Because of the transitivity relations, whenever a new fact is added to a network of interval relations, the consequences of this addition reverberate over the whole network. To compute the full extent of this reverberation would require impractically large amounts of memory, so Allen restricts the 'propagation' of relations to pairs of intervals that share a common reference interval, an idea suggested by Kahn and Gorry's 'special reference events'. For example, if \( i \) has reference interval 1984 and \( j \) has reference interval 1985, then on adding the latter fact there is no need to add also the consequence that \( i \) is before \( j \), since this is easily computed, when needed, from the three facts 'I during 1984', 'J during 1985', and '1984 before 1985' (See Allen, 1981, pp. 223–224).

So far, we have only discussed the treatment of time itself in Allen's system. This would be of little interest for AI without the superstructure which is built upon it, the theory of what happens or obtains in time. Allen uses a threefold ontology of properties, processes, and events; these may be distinguished by the way in which they hold or occur in time. Thus, a property holds over every subinterval of any interval over which it holds (e.g. if I was in Norfolk all last week, then I was in Norfolk all of last Tuesday). For an event, on the other hand, each occurrence defines a unique interval over which it occurs, and it does not occur over any subinterval of that interval. Allen notes the affinity of this treatment of events with McDermott's, but he does not seem to commit the same error of seeking to identify an event with the set of intervals over which it occurs.

Allen's processes sit rather uneasily between properties and events: according to Allen, if I was walking throughout the last hour (Allen says 'over' instead of 'throughout', but it is hard to see what difference this is supposed to make, apart from lending superficial plausibility to what he goes on to claim), then the hour must contain some subintervals throughout which I was walking, but it need not be the case that I was walking throughout every such subinterval. Unfortunately for Allen's theory, it is surely the case that if there is any time during the hour when I was not walking, then it is simply untrue that I was walking throughout the hour; and if one interprets 'walking' broadly so that one can be walking (in this broad sense) throughout a period containing intermittent stretches of not walking (in the narrower sense), then in the broad sense of walking one is walking even during those stretches (in rather the same way that one can be writing a book even though right now one is having a meal, or asleep). For this reason I am unhappy about processes; but I seem to be pretty much alone in this, since most writers on such matters include this category, or something very like it, in their classifications.

Allen uses three different predicates to relate elements from his three ontological categories to the times over which they hold or occur. For a property, he uses the predicate \( \text{HOLDS}(p,t) \) to state that property \( p \) obtains throughout the interval \( t \); for an event he says \( \text{OCCUR}(e,t) \); and for a process, \( \text{OCCURRING}(p,t) \). Allen states axioms for each of these predicates to secure their correct logical behaviour. The axioms for \( \text{HOLDS} \) turn out to be identical with the axioms given by C. L. Hamblin (1971). As I have argued elsewhere (Galton, 1986), these axioms have the weakness that they make it impossible to give a correct treatment of continuous change. This was singled out by McDermott as one of the three most important problems of temporal representation, to which McDermott's own solution was, as already described, to use a continuum of instants as the basis for temporal reference.

Another important divergence from McDermott's system is that Allen rejects the use of a branching future. His ground for this is that 'reasoning about the future is...just one instance of hypothetical reasoning' (Allen, 1984, p. 131). Reasoning about the past is also hypothetical in this way, so Allen would as soon introduce a branching past into his model as a branching future. Note the contrast between the kinds of reason given by McDermott and by Allen for their respective decisions in this matter. Allen is concerned with the structure of our reasoning about time, and argues that in this respect the past and the future are on the same footing. McDermott, on the other hand, is concerned with the way things really are: he is impressed by the thought that the future really is indeterminate whereas what is done is done and cannot be undone. It seems to me that Allen's more pragmatic approach is more appropriate to the kind of enterprise we are considering here; McDermott is in danger of getting himself entangled in a mire of philosophical perplexities—after all, philosophers have argued since the time of Aristotle about the nature of the indeterminacy of the future, and are arguably little nearer to achieving a solution acceptable to all.

One of the major goals of both McDermott and Allen's researches on the representation of temporal reasoning is to facilitate automated planning, particularly in the domain of problem-solving, this being a hallmark of intelligent human activity and thus a prime target for AI. Allen and Koømen (1983) describe a concept of planning whereby a system is presented with initial and goal states and uses these to construct a partial simulation of the
desired future; the planner then looks for ‘causal gaps’ in the simulation and attempts to fill them by the introduction of actions, this procedure being executed recursively until a complete causally connected description of the desired future is achieved.

2.5 Other Recent Approaches

Kowalski and Sergot (1986) have developed a temporal formalism broadly similar to that of Allen, intended for the updating of databases and for understanding narratives. The guiding principle is that since deletion from a database necessarily involves loss of information, it is no good trying to handle change by substituting new facts for old, for then no record will remain of the state of affairs before the change; yet often one needs to know not just how things are now, but also how they were in the past. It is therefore necessary to represent change in the database by explicitly tagging facts with the times at which they are true, and adding new facts as the occasion arises. No more need be said here about the Event Calculus, as Kowalski and Sergot’s system is called, since a detailed treatment is given by Fariba Sadri in Chapter 4 of this volume.

Fariba Sadri also discusses the system of Lee et al. (1985), which employs formalisms adapted from Rescher and Urquhart and from von Wright. Rescher and Urquhart (1971) introduced a notation which can be regarded as a compromise between, on the one hand, the modal style of tense-logical formulae like \( Fp \) and \( Pp \) and, on the other hand, first-order formulae of the form \( p(t) \), where \( t \) stands for a time. Rescher’s notation is

\[ R(t) \]

for ‘\( p \) is realised at \( t \)’. This resembles the first-order notation in explicitly mentioning \( t \), but agrees with the modal approach in respecting the integrity of the proposition \( p \), which may accordingly take any of the forms available within the modal notation, yielding formulae like, e.g. \( R(t) \). In particular, there is no reason why one should not bring in formulae of the kind found in von Wright’s \( T \)-calculus, e.g. \( pTq \), giving us

\[ R(t) \]

Although Rescher himself does not do this, formulae of this type are to be found in Lee et al.’s paper, their notation being \( R(t) \); \( p \& q \). To the extent that Rescher’s notation is partly modal, partly first-order, the same may be said of Lee et al.’s, but the overall impression I get, having worked mainly with modal temporal logics, is that this formalism has a strongly first-order flavour to it, and indeed Fariba Sadri shows that many of the apparent differences between this system and those of Kowalski & Sergot and Allen do not lie more than skin-deep.

To summarize, it is clear that a logical treatment of time has an important part to play in AI, both from the point of view of simulating natural language understanding and from the point of view of simulating intelligent planning. In addition, similar ideas occur in work on databases, which lies outside the sphere of AI proper. Most workers in the field have chosen a first-order formalism, in which times and, in some cases, events are designated by individual terms. It has become apparent that the depth of the conceptual analyses required to bring this endeavour to a successful conclusion is such as to impinge on many of the traditional philosophical problems concerning time, free will, action, etc.; undoubtedly, the computer science community can benefit from the resulting contact with a hitherto rather remote discipline, and equally, as Aaron Sloman has argued quite persuasively, philosophy itself cannot afford to ignore the new computational developments.

3 THE USE OF TEMPORAL LOGIC IN SOFTWARE ENGINEERING

The process of constructing computer programs can be seen as including at least the following three distinct steps:

1. Laying down precisely what the program is to do;
2. Actually writing the program, with the intention that it should satisfy what is laid down in step 1;
3. Checking that the program really does satisfy 1.

These steps may be referred to as specification, synthesis, and validation respectively. As we shall see, temporal logic has been applied to all three steps, beginning with validation. Methods of validating programs may be divided into two distinct types, which we may refer to as testing and verification. Testing a program is a purely empirical matter of actually running it to see whether it works; verification, by contrast, is grounded in theory, and involves mathematically proving that a program is correct. It is to verification rather than testing that temporal logic can be applied.

Before we discuss this in more detail, it is as well to note that, ideally, verification should eliminate the need for testing, but in practice this cannot happen. When one is dealing with a large system, the formal verification becomes exceedingly detailed and cumbersome, and then one’s confidence in the correctness of the proofs may sink to a level comparable to the confidence one might have, prior to any verification or testing, that the program itself has been constructed so as to meet its specifications. In any case, even with a relatively small program, no computer programmer would totally refrain from any empirical tests on the ground that the program had already been
certified correct by theoretical means. So the two approaches have rather come to be seen as complementary.

3.1 Program Verification

How can one ever be sure that a computer program will always do what its designers intended it to do? This is a problem of great practical urgency as well as considerable theoretical interest. In practice, a program is often tested by being ‘put through its paces’, i.e. it is run a number of times with a range of different initial conditions selected so as to take into account all the possible sources of error; if it performs these tests satisfactorily (and of course the stringency of the requirements imposed on the program during testing will depend very much on the likely cost of failure in actual use), then it will be given the seal of approval and let loose upon the world, for better or for worse.

This empirical approach to program certification has the drawback that its credibility depends entirely on the testers’ confidence that they have indeed foreseen all the sorts of ways in which the program might go wrong, and have devised tests accordingly. Since the number of different possible inputs to a program will, in general, be unbounded, while empirical tests can only sample a small finite portion of these, it is not surprising that there has grown up a demand for a more watertight way of validating programs than empirical testing. This demand has led to highly complex theoretical investigations into the concept of program correctness, and it is in this context that temporal logic has made its entry into theoretical computer science.

3.1.1 Floyd’s inductive assertion method

In the early seventies, the dominant paradigm in the theory of program verification was the so-called inductive assertion method, due to Floyd, whose paper ‘Assigning Meaning to Programs’ (Floyd, 1967) is a landmark in the history of this subject. Floyd considered the problem of specifying the behaviour of some simple flowcharts with a view to proving them correct with respect to a given specification. In the following brief account of Floyd’s technique, I shall replace flowcharts by programs with assignments and goto-statements. Thus adapted, the method consists in assigning to each point in a program a propositional tag (known as an invariant), in such a way that it can be proved that if the tag at a given point is true at a time when the program’s control reaches it, then after execution of the command at that point, the tag at the next point reached will also be true. If, now, the tag assigned to the start of the program is true when the program is started up (and this tag can be so devised as to be true so long as the input specification of the program is met), then a simple inductive argument shows that if the program halts then the tag assigned to the end of the program will then be true—so that if this tag is so devised that its truth implies satisfaction of the program’s output specification then it will have been proved that the correct relation holds between the input and output specifications, i.e. that the program is correct (strictly, partially correct—see below).

To illustrate with an extremely simple example, consider the following program, which computes factorials:

\[
\begin{align*}
L_0: & \text{ start} \\
L_1: & f := 1 \\
L_2: & \text{ if } n = 0 \text{ then goto } L_6 \\
L_3: & f := n \times f \\
L_4: & n := n - 1 \\
L_5: & \text{ goto } L_2 \\
L_6: & \text{ end}
\end{align*}
\]

The tags \( T_i \) which are assigned to each program location \( L_i \) by the inductive assertion method are as follows:

\[
\begin{align*}
T_0: & \quad N \in \mathbb{N} \land n = N \\
T_1: & \quad N \in \mathbb{N} \land f = N! \land n = N \\
T_2: & \quad N \in \mathbb{N} \land 0 \leq n \leq N \land f = N! / n! \\
T_3: & \quad N \in \mathbb{N} \land 0 \leq n \leq N \land f = N! / n! \\
T_4: & \quad N \in \mathbb{N} \land 0 < n \leq N \land f = N! / (n - 1)! \\
T_5: & \quad N \in \mathbb{N} \land 0 < n < N \land f = N! / n! \\
T_6: & \quad N \in \mathbb{N} \land n = 0 \land f = N!
\end{align*}
\]

This example illustrates the three key programming constructs covered by Floyd’s account: conditional branching, join of control, and assignment. At a conditional branch of the form

\[
L_i: \text{ if } \phi \text{ then goto } L_j \text{ else goto } L_k,
\]

the following relations must hold between the tags:

\[
\begin{align*}
T_j & \equiv T_i \land \phi \\
T_k & \equiv T_i \land \neg \phi
\end{align*}
\]
It is easily seen that these are indeed the relations holding amongst \( T_2, T_6, \) and \( T_4 \) in our example.

Again, where two control-paths merge, the rule is that if \( L_i \) and \( L_j \) both lead to \( L_4 \), then

\[
T_k = T_i \lor T_j.
\]

In our example, this is precisely the relation holding between \( T_1, T_3, \) and \( T_2 \).

Finally, the rule for tagging assignment statements is a little more complicated when stated in generality, but easy enough to follow in our example. The assignment \( f := n \times f \) at \( L_3 \) clearly converts \( T_3 \) to \( T_4 \), since

\[
n \times N! \times n! = N! / (n - 1)!
\]

while the next assignment \( n := n - 1 \) affects both conjuncts of the tag in an obvious way.

That the set of tags shown above constitutes a formal verification of our factorial program follows from the fact that the tags have been constructed in accordance with the correct rules governing tagging for the relevant program constructs; what has been proved is that when the program is run with input \( n = n_0 \), where \( n_0 \) is a positive integer, then if it ever reaches the terminal location \( L_4 \), the values of the program variables at that point will be \( n = 0 \) and \( f = n_0! \). And since the program is intended to compute factorials, we can thus assert that if it terminates, it will do so with the intended result.

What the inductive assertion method does not prove is that the program will terminate. To prove termination, Floyd had to use a completely different method, whereby the program locations are now tagged, not with assertions, but with functions of the program variables whose values are taken from some well-ordered set (typically, the set of natural numbers, or the Cartesian product of this set with itself some number of times). The tagging must be carried out in such a way that during execution of the program the value of the tag, with the current state of the program variables as arguments, steadily decreases with respect to the well-ordering. Given this condition, any possible execution-sequence of the program will correspond to a strictly descending sequence of terms from the well-ordering; from the definition of a well-ordering, any such sequence must be of finite length, and hence so must all execution-sequences of the program, i.e. the program must terminate.

### 3.1.2 Hoare's axiomatization of the inductive assertion method

The chief importance of Floyd's paper lies in the introduction of a set of rules governing the construction of invariants (tags). Hoare (1969) further systematized this idea by formulating such a set of rules as a rigorous

axiomatic system, thereby importing the methods of formal logic into the enterprise of reasoning about programs.

Hoare adopted a basic schema of the form \( [P]S[Q] \); to be interpreted as meaning that if the assertion \( P \) is true when the program \( S \) is initiated, then the assertion \( Q \) will be true if and when it terminates. Hoare used this formalism to state rules specifying the desired behaviour of each construct in his programming language. In his initial paper, the constructs treated are just assignment, sequential composition, and the while-statement.

Hoare's system, like any other formal calculus, consists of a set of axioms together with a set of rules of inference for deriving theorems from the axioms. In fact, Hoare uses only a single axiom, the Axiom of Assignment:

\[
\vdash [P[a/x]] \ X := a \ [P] \quad \text{(Ass)}
\]

Here \( [P[a/x]] \) denotes the proposition obtained from \( P \) by replacing each occurrence of 'a' in \( P \) by 'a'. In addition, there are the following rules of inference:

(i) Rules of Consequence

\[
\begin{align*}
&\vdash [P]S[Q], \text{ and } Q \implies R, \text{ then } \vdash [P]S[R] \\
&\text{ (Cons 1)}\\
&\vdash [P]Q, \text{ and } \vdash [Q]S[R], \text{ then } \vdash [P]S[R] \quad \text{ (Cons 2)}
\end{align*}
\]

(ii) Rule of Composition

\[
\begin{align*}
&\vdash [P]S_1[Q] \text{ and } \vdash [Q]S_2[R], \text{ then } \vdash [P]S_1; S_2[R] \quad \text{(Comp)}
\end{align*}
\]

(iii) Rule of Iteration

\[
\begin{align*}
&\vdash [P \land B]S[P] \text{ then } \vdash [P] \text{ while } B \text{ do } S \{ \neg B \land P \} \quad \text{(It)}
\end{align*}
\]

To illustrate how Hoare's system works, we cannot use exactly the same program as we used to illustrate Floyd's method, because Hoare's logic does not deal with goto-statements. Instead, we shall paraphrase our factorial program as a while-program, as follows:

\[
\begin{align*}
\text{begin} \\
&f := 1; \\
&\text{while } n > 0 \text{ do} \\
&\begin{align*}
&f := n \times f; \\
&n := n - 1;
\end{align*} \\
\text{end.}
\end{align*}
\]
Let this program be $S$, and let the block within the while-loop, i.e. 'begin $f := n * f; n := n - 1$; end', be $S_1$.

A formal proof of the partial correctness of $S$ can now be written thus:

1. \( \{ n = N \land 1 = 1 \} \ f := 1 \ \{ n = N \land f = 1 \} \) \quad \text{[Ass]} \[1, \text{Cons} 2 \]
2. \( \{ n = N \} \ f := 1 \ \{ n = N \land f = 1 \} \) \quad \text{[1, Cons 2]} \[1, \text{Cons} 2 \]
3. \( \{ 0 < n \leq N \land n * f = N!/(n-1)! \} \ f := n * f \ \{ 0 < n \leq N \land f = N!/(n-1)! \} \) \quad \text{[Ass]} \[3, \text{Cons} 2 \]
4. \( \{ 0 < n \leq N \land f = N!/(n-1)! \} \ f := n * f \ \{ 0 < n \leq N \land f = N!/(n-1)! \} \) \quad \text{[3, Cons 2]} \[3, \text{Cons} 2 \]
5. \( \{ 0 < n \leq N \land f = N!/(n-1)! \} \ n := n - 1 \ \{ 0 < n < N \land f = N!/(n-1)! \} \) \quad \text{[Ass]} \[5, \text{Cons} 2 \]
6. \( \{ 0 < n \leq N \land f = N!/(n-1)! \} \ n := n - 1 \ \{ 0 < n < N \land f = N!/(n-1)! \} \) \quad \text{[5, Cons 2]} \[5, \text{Cons} 2 \]
7. \( \{ 0 < n \leq N \land f = N!/(n-1)! \} \ \text{while} \ n > 0 \ \text{do} \ \{ \neg (n > 0) \land 0 \leq n < N \land f = N!/(n-1)! \} \) \quad \text{[8, It]} \[8, \text{Cons} 2 \]
8. \( \{ 0 < n \leq N \land f = N!/(n-1)! \} \ \text{while} \ n > 0 \ \text{do} \ \{ n = 0 \land f = N! \} \) \quad \text{[9, Cons 1]} \[9, \text{Cons} 1 \]
9. \( \{ n = N \land f = 1 \} \ \text{while} \ n > 0 \ \text{do} \ \{ n = 0 \land f = N! \} \) \quad \text{[10, Cons 2]} \[10, \text{Cons} 2 \]
10. \( \{ n = N \land f = N! \} \ \text{while} \ n > 0 \ \text{do} \ \{ f = N! \} \) \quad \text{[2, Cons 1]} \[11, \text{Cons} 2 \]
11. \( \{ n = N \land f = 1 \} \ \text{while} \ n > 0 \ \text{do} \ \{ n = 0 \land f = N! \} \) \quad \text{[2, Cons 1]} \[11, \text{Cons} 2 \]
12. \( \{ n = N \} \ S \ \{ n = 0 \land f = N! \} \) \quad \text{[2, Cons 1]} \[12, \text{Cons} 1 \]
13. \( \{ n = N \} \ S \ \{ f = N! \} \) \quad \text{[2, Cons 1]} \[12, \text{Cons} 1 \]

Note that in this proof every assertion enclosed within braces ought to have an additional conjunct 'n ∈ N', which I have omitted for the sake of clarity. These extra conjuncts ensure that the rules Cons 1 and Cons 2 work correctly: for example, in passing from line 5 to line 6, we make use of the implication:

\[ 0 \leq n - 1 < N \Rightarrow 0 < n \leq N. \]

The validity of this implication can only be guaranteed so long as 1, n, and N belong to a data type for which appropriate logical rules hold. The set N of natural numbers constitutes one such data type, being, of course, the domain of the intended interpretation of the program. A proper axiomatic specification of the data types used by a program is thus an essential part of Hoare's method.

Hoare's paper, though brief and lacking in detail, has been enormously influential. Hoare himself remarked that we should accept the axiomatic proof theory for a programming language as 'the ultimately definitive specification of the meaning of the language'. The idea that the proof system gives the meaning of a programming language, and hence of programs themselves, was already explicit in the title of Floyd's paper, and has led to this whole enterprise becoming known as axiomatic semantics.

Let us note in passing that in this use of the term ' semantics' there is a curious inversion of the usual use of that term in logic. Ordinarily, the formal semantics of a proof system is given by relating it to a model theory defined in set-theoretic terms. In the present instance, a proof system (Hoare's logic) is used to specify the semantics of something which is not a proof system, namely a programming language (or, more precisely, a class of implementations thereof). Even so, there is still present in this use of the term ' semantics' the all-important idea of a systematic correspondence between the entities whose intended meanings are to be formally specified and some other class of entities whose formal properties can be independently given.

This being so, the usual metatheoretical questions regarding soundness and completeness of the axiomatization arise in connection with programming language semantics too, and in particular, if the axiomatic semantics is to be of practical use to actual program-users, it is obviously important to determine precisely what class of language implementations a given Hoare-style axiom system is applicable to (cf. Bergstra and Tucker, 1984). Otherwise, for example, one could not be sure that a program which has been certified as correct when run on one implementation of its language will still be correct when run on another implementation.

As originally presented, Hoare's logic dealt with only a very limited set of programming language constructs. Over the years following its introduction, Hoare's ideas have been extended by numerous workers to cover additional constructs such as procedures, recursion, and goto-statements (see Apt, 1981, for a survey, and de Bakker, 1980, for a rigorous treatment of many different constructs).

Of particular importance for our present concerns is the attempt by S. Owicki and D. Gries (1976) to adapt the Floyd-Hoare technique to reasoning about parallel or concurrent computation. With hindsight, it must be admitted that the main conclusion to be drawn from this work is that Hoare's logic is not the most suitable tool for specifying the meanings of parallel programs, and one outcome of this is a proliferation of attempts to find alternative, superior methods (see Barringer, 1985, for a survey).

It is against this background that the temporal approach to program
verification to be discussed in the ensuing sections must be understood. This development is a natural consequence of the vastly increased emphasis on the problem of theoretical program verification initiated by the Floyd-Hoare approach, and on the use of logical tools for its solution.

3.1.3 A new approach: Burstall’s intermittent assertion method

There is something displeasing about having to use two utterly different methods, one to prove that a program terminates, the other to prove that if it terminates it does so correctly. The latter condition is known as partial correctness; the two conditions together constitute total correctness. Burstall (1974) devised a method whereby total correctness could be secured in a single proof. Burstall described his method as 'hand simulation with a little induction'. The idea is that one follows through the execution of the program by hand, using symbolic data instead of actual numbers, invoking mathematical induction to prove general statements about what happens at a loop point. As Burstall put it, the prover acts as 'a sort of symbolic interpreter, with a "state vector" giving symbolic expressions instead of numbers as identifier values' (Burstall, 1974, p. 308).

To illustrate the method, let us turn again to our original factorial program. The statement to be proved, i.e. total correctness, can be expressed as

$$\text{For all } N \in \mathbb{N}; \text{ at } L_0 \Rightarrow \text{ sometime (at } L_4 \land f = N!).$$

To begin the proof, we assume that the program is at some time at $L_0$ with $n = N \in \mathbb{N}$. From $L_0$, control can only pass through $L_1$ to $L_2$; after the assignment at $L_1$ we thus have

$$\text{at } L_2 \land n = N \land f = 1$$

This reaches a loop point, so we must use induction. What we prove is that

$$\text{for all } i \text{ with } 0 \leq i \leq N, \text{ we eventually have}$$

$$\text{at } L_2 \land n = N - i \land f = N!/n!$$

The base case is straightforward: it is (15) above. For the induction step, assume $0 \leq i < N$ and (16). Since $n = N - i > 0$, control must next pass through $L_3, L_4, L_5$ and back to $L_2$. Computing the effect of the assignments at $L_3$ and $L_4$ gives us

$$\text{at } L_2 \land n = (i + 1) \land f = N!/n!$$

which is the inductive hypothesis for $i + 1$, as required.

Finally, for $i = N$, we have

$$\text{at } L_2 \land n = 0 \land f = N!$$

and since $n = 0$ the next step is $L_6$, giving us

$$\text{at } L_6 \land f = N!$$

as required.

It is important to note that what we have proved here is not just a conditional, that if we get to $L_6$, then $f = N!$, but the categorical assertion that we will get to $L_6$ with $f = N!$.

Burstall's paper demonstrated the new method on several programs, most of them considerably more complex than our example. The final section of the paper, entitled 'A connection with Modal Logic', is the most significant for our present concerns, for in it Burstall contrasts the implicit form of the statements used in his own proofs, namely

$$\text{sometime (at } L \land \ldots),$$

with the form of statement implicit in Floyd's assertions, namely

$$\text{always (at } L \rightarrow \ldots),$$

and notes that both types of statement can be seen as belonging to a simple modal logic. This appears to be the first suggestion that a form of modal logic might be useful in reasoning about programs. Burstall ends with the telling remark that 'further investigation... might be profitable, asking "what kind of modal logic underlies our informal arguments about program executions?".'

3.2 Further Developments of the Modal Approach

Manna and Waldinger (1978) characterized the Floyd-type assertions of the form 'always(at $L \rightarrow \phi$)' as invariant assertions, and the Burstall-type ones of the form 'sometime(at $L \land \phi$)' as intermittent assertions. They showed that the intermittent-assertion method is at least as powerful as, and never less simple to use than the invariant-assertion method. In particular, they showed how an invariant-assertion proof can always be converted, in a more or less mechanical way, into an intermittent-assertion proof.

Pnueli (1977) explicitly systematized the modal logic adumbrated by Burstall as a temporal logic with future-tense operators $\Box$ (read 'sometime' or 'eventually') and $\square$ (read 'always') corresponding to Prior's $F$ and $G$ respectively. The correspondence is not quite exact, in that Pnueli's operators receive an interpretation in which the present counts as part of the future, so that for $\square \phi$ to be true now, it is necessary not only that $\phi$ should be true at all future times, but also that it should be true now; and for $\Box \phi$ to be true now, it suffices that $\phi$ itself should be true now, without necessarily being true at any
future time. The semantics appropriate for temporal logic in the context of program verification is of course based on discrete time, this corresponding to the discrete structure of the execution-sequence. As a result, Pnueli and many others have found it helpful to include in their language an additional temporal operator $\Diamond$, such that $\Diamond \phi$ is true at a given time just if $\phi$ is true at the next time.

The other distinctive feature of Pnueli's system is that it presents a logic of future time only. This is because the kind of reasoning about programs that is formalized by means of this logic always works forward from a given program-state to the states which succeed it in the execution-sequence. This too has repercussions on the style of the formal semantics. The Kripke semantics for normal tense logic with past and future operators interprets formulae on structures of the form $\langle T, R, t \rangle$, where $T$ is a set of times ordered by the relation $R$, and $t$ is a member of $T$ singled out as the time at which the formula is to be evaluated. The set $T$, as ordered by $R$, may be dense or discrete, bounded or unbounded, linear or branching. In Pnueli's system, on the other hand, since the temporal formulae are to be used for reasoning about execution-sequences, the structures over which the semantics interprets them must themselves be formal representations of execution-sequences. This constrains $\langle T, R \rangle$ to be a discrete linear ordering isomorphic to the set of natural numbers; and it is no longer necessary to include a third component, $t$, in the model structures, since it is always tacitly understood that formulae are evaluated on the first term of an execution-sequence. To evaluate a formula on a later term in a sequence, one considers instead the sequence obtained by chopping off a prefix of the original sequence so that the desired term becomes the first term in the new sequence.

It soon became apparent that the new formalism was a useful tool not just for proving program correctness but also for reasoning about programs generally, and that it was particularly suitable for reasoning about concurrent programs (in which several processes are executed in parallel, thus as it were competing for the attention of the central processor and requiring to be correctly coordinated in time) and cyclic or reactive programs (such as operating systems, which do not terminate but rather maintain a continual interaction with their environment). For these tasks, it is of course important to have good abstract models of concurrency, with the result that the application of temporal logic has been an important stimulus towards the development of such models (see e.g. Manna and Pnueli, 1981).

In the next two sections, we shall review the further development of temporal logic in this area of computer science. We shall begin by considering developments in the formal language and its interpretation (syntactic and semantic developments), and then take a look at the wider context of its use in reasoning about programs (pragmatic developments).

### 3.3 Syntactic and Semantic Developments

It is not possible to keep syntactic and semantic issues sharply separate; this is because syntactic innovations are generally made with a view to increasing, or at least modifying, the expressive power of a language, and expressive power is not a property of syntax alone, but of syntax in relation to a given semantics.

#### 3.3.1 Linear and branching time

A good example of the interdependence of syntax and semantics is furnished by the divergence that has appeared between systems based on linear-time models and those based on branching time. In order to allow for the possibility of interpretations in which the set of times has a branching structure, Lamport (1980) distinguished between tense operators with the meanings 'sometime' (symbolized $\rightarrow$) and 'not never' ($\Diamond$), the latter being defined in terms of 'always' ($\Box$) as $\neg \Diamond \neg$.

In Lamport's branching-time structure, $\Box p$ is interpreted to mean that $p$ is true throughout every possible future; $\Diamond p$ accordingly means that $p$ is true at some time in some possible future; while for $\rightarrow p$ the interpretation is that $p$ is true at some time in every possible future. Clearly $\Diamond p$ and $\rightarrow p$ are not equivalent in this interpretation; but if we reduce the number of possible futures at each time to one, thereby effectively reverting to a linear-time model, $\Diamond p$ and $\rightarrow p$ do come out as equivalent.

Now Lamport was particularly concerned to establish the relative merits of the two kinds of interpretation in relation to different purposes. He noted that in proving many properties of concurrent programs, a commonly used type of reasoning makes implicit appeal to the principle that any proposition $p$ is either always false or eventually true; and this principle, according to Lamport, implies an axiom of the form

$$\rightarrow p \lor \Box \neg p$$

Since this axiom is valid for all models under the linear-time interpretation, but not under the branching-time interpretation, Lamport concluded that linear time was the more appropriate framework for formalizing this kind of reasoning.

Emerson and Halpern (1983) pointed out the flaw in Lamport's argument: the axiom (17) is only a correct formalization of the principle in question so long as the linear-time interpretation is assumed. In effect, Lamport had assumed that (17) expresses the same principle in both linear and branching time. But in branching time, the relevant version of the principle is that $p$ is
always false or eventually true for every possible future taken individually; and this principle cannot be expressed in the language used by Lamport.

The required extension to the language was in fact not yet available in 1980, when Lamport's paper appeared. By the time of Emerson and Halpern's paper, there was already in existence a series of branching-time logics of varying expressive power, including some in which it was indeed possible to express the desired version of Lamport's principle.

The crucial innovation underlying the development of this series was due to Ben-Ari et al. (1981). They developed a language UB (for 'unified system of branching time') in which the temporal modalities combine quantification over possible futures with quantification over individual times within a future. This gives rise to six modalities, which may be written (here diverging slightly from Ben-Ari et al.) as $\forall\Box$, $\forall\Diamond$, $\exists\Box$, $\exists\Diamond$, $\forall\circ$ and $\exists\circ$. The interpretation of these symbols should be clear from, e.g.,

$\exists\Box p$ is true at $t$ iff $p$ is true throughout some possible future of $t$;

$\forall\Diamond p$ is true at $t$ iff $p$ is true at the next time in every possible future of $t$.

In UB, these symbols are not decomposable: that is, $\forall$ and $\exists$ can only occur immediately preceding $\Box$, $\Diamond$, or $\circ$; and members of the latter set can only occur immediately following $\forall$ or $\exists$.

The branching-time version of Lamport's principle cannot be expressed in UB; but it can be expressed if we allow the path-quantifiers $\forall$ and $\exists$ to be prefixed to assertions composed of arbitrary combinations of tense-operators. The required formula is then:

$$\forall(\Diamond p \lor \Box \neg p)$$

(18)

In effect, this says that (17) holds in every possible future, as required.

Emerson and Halpern discuss a range of branching-time languages, and nicely characterize their syntactic structures by drawing a distinction between state formulae, which describe what is the case at individual times, and path formulae, which describe what holds over an entire future path. Our example (18) above is a state formula since an assertion about all possible futures from a given time is an assertion about that time rather than about any one possible future; but what is asserted at each of these futures, namely (17), is an assertion about a single entire future, and thus a path formula. So we see that $\forall$ has the effect of converting path formulae to state formulae. On the other hand, $\Box$ and $\Diamond$ convert state or path formulae into path formulae.

Emerson and Halpern give a list of eleven such conversion rules which together define a language they call CTL*. By selecting appropriate subsets of this set of rules, one can give exact definitions of the other branching-time logics that have been proposed at various times by different authors.

Applying this form of analysis to the system UB discussed above, we see that this system lacks the rule that if $p$ and $q$ are path formulae, so is $p \lor q$; in this language, $\Diamond p$ and $\Box \neg p$ are path formulae, so there exist state formulae $\forall\Diamond p$ and $\forall\Box p$; but because the disjunction of path formulae is not again a path formula, $\Diamond p \lor \Box \neg p$ is not a path formula, so there is no state formula $\forall(\Diamond p \lor \Box \neg p)$. Another branching-time logic derivable in this way is CTL (Computation Tree Logic) of Emerson and Clarke (1982). CTL is similar to UB, but unlike the latter includes the 'until' operator $U$ in its formalism. It is mentioned here because we shall have cause to refer to it again below.

3.3.2 Fairness and related properties

Another syntactic innovation, which figures in both linear and branching-time systems, is also closely linked with the issue of expressive power. As we noted above, the use of temporal logic in reasoning about programs need not just be confined to proving correctness. In concurrent programs, an important class of properties are those relating to fairness, i.e. the requirement that two or more concurrent processes have as it were equal rights of access to the central processor. This requirement needs to be stated explicitly because from the point of view of the model of concurrency used in the temporal framework the choice as to which process is scheduled at each point in the execution is to all intents arbitrary, being determined in practice by extraneous features to do with the specific implementation of the program; hence fairness cannot be guaranteed a priori but must be demonstrable as a program property like any other.

Fairness is not, in fact, a single property, but covers a range of different possible requirements. A weak requirement, for instance, is that a process which is continuously active will eventually be scheduled (this property is called justice by Lehmann et al., 1981); a stronger requirement is that a process which is active infinitely often will be scheduled; stronger still is that a process which is active just once will be scheduled. These properties have to do with how much prompting by a process the control of the program needs before it will respond by allocating time to it. Because of this, these properties are called responsiveness properties by Gabbay et al. (1980), who reserve the term 'fairness' for requirements such as that, of two processes, the one that is active sooner will be scheduled sooner (strict fairness).

Responsiveness properties can be expressed using the temporal logic based on $\Box$ and $\Diamond$, but for strict fairness the language must be extended. Gabbay et al. introduce for this purpose a binary operator $U$, read 'until', such that $vUw$ is true just now so long as $v$ will be true at all times until some future time when $w$ will be true. This operator is, of course, one of the pair first introduced by Kamp (see above, p. 11), and Kamp's result, that the temporal
logic based on this pair of operators is expressively complete with respect to the first-order properties of a certain class of models, carries through to the logic of Gabbay et al., in which only the future fragment of the full temporal logic is considered. One of the main aims of their paper is to present a simpler proof of this result for that future fragment alone.

Program properties requiring $U$ for their expression are called precedence properties (Manna and Pnueli, 1981), in contrast to invariance properties, which just require $\square$, and eventuality properties, which require $\Diamond$ (Lamport, 1980, calls these latter properties safety and liveness properties respectively).

Subsequently, a weaker version of $U$, somewhat misleadingly read 'unless', which does not imply eventual realization of the second argument, has also been found useful. This weaker operator has been variously noted; following Barringer (this volume, Ch. 2), we shall use $W$. The operators $U$ and $W$ are mutually interdefinable by means of the equivalences

$$pWq \equiv pUq \lor \square p$$
$$pUq \equiv pWq \land \Diamond q.$$

### 3.3.3 Extended temporal logic

I remarked above that expressive power is a property of interpreted languages, i.e. languages endowed with a semantics as well as a syntax. Expressive completeness is a relation involving a further term, since to call a language expressively complete is to say that one can express in it all the properties of some given class. So expressive completeness is, properly speaking, a property that an interpreted language holds in relation to a class of properties, these being, of course, properties of its models.

The sense in which Gabbay et al. showed temporal logic with until to be expressively complete is that it can be used to express all first-order properties of the execution-sequences of programs. But as Wolper (1981) pointed out, not all properties are first-order properties; Wolper’s example is the property that a certain proposition is true at every nth step of the computation (and possibly at other steps too). This property cannot be expressed in temporal logic with until, yet as Wolper remarks, one might wish to use it in reasoning about a program which has to check for the satisfaction of some condition every nth execution step.

To get round this difficulty, Wolper proposed a method of adding new operators to temporal logic in such a way that its expressive power becomes equal to that of any given right-linear grammar. A right-linear grammar consists of a finite set of production rules of the form

$$V_i \rightarrow v_1 \ldots v_j V_k$$

where the $V_i$ are non-terminal symbols, and the $v_i$ terminal symbols of the grammar. Expressions generable by these rules are called regular expressions.

The connection with temporal operators may be illustrated by the grammar

$$V \rightarrow q$$
$$V \rightarrow pV.$$

This grammar (Wolper’s Example 2, p. 343) generates all sequences of the form

$$q, pq, ppq, pppq, ppppq, \ldots,$$

as well as an infinite sequence of $ps$; and these are precisely the ways in which an execution-sequence must begin in order for it (or its first term, in a point-based semantics) to satisfy the temporal-logic formula $pWq$. The link between the grammar and the temporal operator $W$ is that the grammar corresponds, in a systematic way which is expounded in full generality by Wolper, to the following axiomatization of the operator:

$$\forall pWq \rightarrow q \lor (p \land \Diamond (pWq))$$
$$\forall u \land \square (u \rightarrow q \lor (p \land \Diamond u)) \rightarrow pWq.$$

Now the point is that any right-linear grammar gives rise by way of such an axiomatization to a new temporal operator. In particular, it is possible to write a grammar which generates all and only the sequences with $p$ true on every $n$th step and arbitrary terms elsewhere; Wolper’s method then automatically converts this grammar into an axiomatization of a new temporal operator, thus enabling one to express the corresponding property of execution-sequences in the resulting temporal logic. So to each right-linear grammar there corresponds an Extended Temporal Logic (ETL), which is expressively equivalent to the grammar. Of course, no one such grammar subsumes all the rest, so this method will certainly not give us a logic that is expressively complete with respect to the totality of regular expressions; rather, what Wolper gives us is a way of extending temporal logic to cover specific classes of regular expressions as required.

### 3.3.4 Interval semantics

All the systems considered so far have a semantics based on either individual times or whole futures. In linear time, these are equivalent, since there is a one-to-one correspondence between times and their (unique) futures: it makes little difference whether we say that a formula like $\square p$ refers to a property of times (viz. that $p$ is true at every moment from a given time on) or
to a property of futures (viz. that \( p \) is true throughout a given future). In branching-time models, though, there is no longer this straightforward equivalence, since each time has many possible futures; in this case, then, semantics based on individual times and semantics based on whole futures have different parts to play in the process of building up the interpretation of a formula from basic elements, point-based semantics being appropriate for state formulae, future-based for path formulae.

A radical innovation due to Moszkowski (1983) is to base the semantics on intervals. The motivation for this is to facilitate reasoning about finite chunks of program behaviour, as distinct from entire execution sequences. This kind of reasoning has also been applied to computer hardware. The innovation at the semantic level naturally gives rise to syntactic novelty too, including for example the operator \( \cdot \) (read 'chop'), with the interpretation that \( pq \) is true on an interval just so long as that interval can be decomposed into two contiguous subintervals in such a way that \( p \) is true on the earlier of the two, and \( q \) on the later. There is nothing corresponding to this operator in languages whose semantics is point-based rather than interval-based; indeed, it is hard to see how there could be, since there is no way of decomposing a point into two component points. Further details of Moszkowski’s system ITL (Interval Temporal Logic) can be found in Roger Hale’s contribution to this volume (Chapter 3).

### 3.4 Pragmatic Developments

Under the heading of pragmatics are included all those features of a language which go beyond the mere formal description of its syntax and semantics, relating rather to the broader context of how the language is used. The distinction is a somewhat artificial one and cannot do full justice to the fact that syntax, semantics and pragmatics interact in ways which make it impossible to draw sharp boundaries; but taking them as approximate general headings it is clear enough what sort of features are to be included under each.

To help us understand the main pragmatic developments, let us first consider the general question of why temporal logic is such a suitable tool for reasoning about programs. The answer lies in the fact that programming languages and temporal logics can both be thought of as having to do with the description of sequences. The operational semantics of a programming language is, in effect, a means by which the set of possible execution-sequences of a program can be derived from the program itself. In parallel with this, the model-theoretic semantics of a temporal logic relates formulae of the logic to the sets of models which satisfy them, and typically these models also take the form of sequences. Identifying the latter sequences with

the execution sequences, temporal formulae and computer programs are brought into relation with a common set of structures, and by means of this indirect relationship a direct relation is established between programs and temporal logic formulae, namely that a set of temporal formulae \( \Sigma \) can be said to specify a program \( P \) just so long as every execution-sequence of \( P \) is a model for \( \Sigma \) (see Fig. 1). In this way the programming language can be said to be endowed with a temporal semantics.

In the temporal semantics of Manna and Pnueli, the aim is to use temporal logic to prove that a program satisfies certain specifications—namely the statement of its correctness, together with other features like fairness which are involved in the notion of correctness of concurrent programs. The proof takes as its premisses certain temporal formulae which by inspection are obviously satisfied by the program. Temporal logic is then used to deduce other formulae which are less obviously satisfied by the program; that they nonetheless must do so is a result of the pre-established harmony between the model-theory of the temporal language and the operational semantics of the programming language.

As described, this method suffers from a number of serious drawbacks. For one thing, it can only be applied to already-existing complete programs; for another, it generally requires a lot of detailed and tedious working in all but the simplest cases. The developments we shall consider in this section are all aimed at improving this situation.

First, we shall consider attempts to simplify the proof stage by automating the process of deduction. Next, we shall look at attempts to synthesize programs from the temporal logic specifications, thus getting over the drawback inherent in Manna and Pnueli’s method that programs must be given in advance. Third, we shall discuss work directed towards modularizing the proof process, so that instead of having to be applied to complete

![Fig. 1.](image-url)
programs, temporal reasoning can be applied to smaller program modules in such a way that the reasoning carries over from the separate modules to any larger program of which they are part. Finally, we shall examine attempts to simplify the scheme of Fig. 1 by unifying the temporal specification language with the programming language.

### 3.4.1 Automatic verification

The goal here is to be able to show automatically that the temporal logic formula expressing a program’s correctness is a logical consequence of some set of formulae immediately derivable from the program itself. An obvious way to set about achieving this is to try to construct an automatic theorem-prover for the temporal logic in question.

Theorem-proving for ordinary first-order logic generally makes use of resolution (Robinson, 1965), so a natural line of inquiry is to look for temporal analogues of the resolution principle. This line has been followed up by Abadi and Manna (1985), who use non-clausal resolution, in which the formulae are not first required to be cast into clausal form, and by Farinas del Cerro (1985), who uses a form of clausal resolution analogous to the clausal resolution principle for first-order logic. These authors have had a certain degree of success in devising workable methods, but it is not yet clear whether it will prove practicable to implement them efficiently for the purposes of automatic program verification.

An alternative approach, due to Clarke et al. (1986), is to exploit the model theory of temporal logic rather than its proof theory. The idea is that instead of trying to verify a concurrent system by deducing a formula which expresses the system’s correctness from other temporal formulae, we can check directly whether the state-graph of the concurrent system, considered as a Kripke model, satisfies the formula. To do this, of course, requires that the state-graph be finite, so only finite-state systems can be handled in this way; but this is not, the authors claim, too severe a limitation.

Clarke et al. use CTL for their specification language (see above, p. 39). The use of a branching-time logic reflects the fact that the totality of execution-sequences, obtainable as the set of possible paths through the state graph, can be viewed as a branching structure. The model-checking algorithm has complexity which is linear in both the size of the specification and the size of the state-graph. This low complexity compares favourably with what could be achieved using a similar method for linear-time temporal logic.

### 3.4.2 Automatic synthesis

In this section we consider systems that have been devised to synthesize the synchronization part of a concurrent program, that is, the part of the program concerned with securing the correct relative timing of the execution of the constituent processes. The synchronization part in most concurrent programs can be handled separately from the functional part, which does all the actual computation; and of course it is mainly in reasoning about the synchronization part that temporal logic has proved useful.

Manna and Wolper (1984) use a linear-time propositional temporal logic to specify the individual processes, modifying the resulting specifications in order to reflect the embedding of each process in the environment constituted by all the others, and then they apply a satisfiability algorithm, derived from the tableau method of Rescher and Urquhart (1971), to obtain a state-graph for the program. After suitable modifications, the state-graph can be used to derive the synchronization part of the program in the form of usable code.

The model of concurrency used by Manna and Wolper is Hoare’s Communicating Sequential Processes (Hoare, 1978), in which a collection of sequential processes interact via input/output operations and in no other way. Emerson and Clarke (1982) use a different model of concurrency, in which the constituent processes can interact by having common access to a shared memory. In order to synthesize the synchronization part (synchronization skeleton in their terminology), Emerson and Clarke use CTL rather than linear temporal logic. As with Manna and Wolper, the synthesis proceeds by building up a tableau which can be read as a state-graph for the concurrent system.

Emerson and Clarke defend their use of branching-time logic on the ground that it enables them to assert the existence of computation paths having specified properties. This, they say, can be helpful in ensuring that the synthesized program is not a ‘degenerate’ solution with only a single path. Manna and Wolper, while admitting this, do not regard it as a useful feature, since one is really only ever interested in those properties which hold for all computations of a program. Instead, they extol the greater simplicity of linear temporal logic, and the possibility of extending it by the addition of Wolper’s ‘grammar operators’ (see above, p. 41). As things stand, it is not yet clear which of the two approaches will win the day, or if indeed both might have a part to play in the future development of concurrent programs.

### 3.4.3 Compositional specification

Barringer et al. (1984) pointed out that existing temporal logics for program verification suffered from the serious drawback that they could only be applied to complete programs already in existence; whereas what was required for the rigorous systematic development of concurrent programs was a way of using temporal specifications to build up programs bit by bit, specifying and verifying each part, or module, separately, and combining the
separate specifications to produce a specification of the whole program: in short, what was required was a modular approach, in contrast to the global approach prevalent up till then.

In order to do this, Barringer et al. have developed a compositional system of concurrent program specification using temporal logic. The guiding idea is that the temporal specification language should mirror the language of the program in sufficient detail for the modular structure of the program to be expressible in the specification language. In terms of Fig. 1, what is needed is to make the diagram more nearly symmetrical, with each construct in the programming language (e.g. assignment, concatenation, conditional statement, while-statement, parallelism) being given expression in the temporal logic. Explicit appeal to execution-sequences is avoided by suitably axiomatizing the relationship that holds between the programming language and the temporal logic.

To give just one simple example, consider the construct of sequential composition or concatenation, whereby two program statements $S_1$ and $S_2$ are combined into a new statement $S_1; S_2$. The operational semantics for concatenation asserts that for $\sigma$ to be an execution-sequence for $S_1; S_2$, $\sigma$ must be decomposable as $\sigma_1 \cdot \sigma_2$, where $\sigma_1$ and $\sigma_2$ are execution-sequences for $S_1$ and $S_2$ respectively. The symbol $\sigma_1 \cdot \sigma_2$ here represents the fusion of $\sigma_1$ and $\sigma_2$, the execution-sequence obtained by appending $\sigma_2$ to the end of $\sigma_1$, if the latter is finite, or else identical to $\sigma_1$, if it is infinite. Corresponding to concatenation in the programming language, a new binary combination operator $C$ is introduced into the temporal specification language. The semantics of $C$ specifies that an execution-sequence $\sigma$ satisfies $\phi_1 C \phi_2$ just so long as it can be decomposed as $\sigma_1 \cdot \sigma_2$, where $\sigma_1$ and $\sigma_2$ satisfy $\phi_1$ and $\phi_2$ respectively.

It is clear from the similarity of the two definitions, of concatenation and combination, that the satisfaction relation thereby set up between program statements and temporal specifications is the right one. The axiomatic definition takes the form of the following rule of inference: if $S_1$ satisfies $\phi_1$, and $S_2$ satisfies $\phi_2$, then $S_1; S_2$ satisfies $\phi_1 C \phi_2$. Thus a correct specification for $S_1; S_2$ can be obtained directly from correct specifications of $S_1$ and $S_2$.

In this way, the temporal logic specification of a complex program can be built up from the specifications of its separate components. This technique is still in its infancy; Howard Barringer's contribution to this volume (Ch. 2) presents the current state of thinking in this area.

3.4.4 Temporal logic as a programming language

In all the applications we have considered so far, temporal logic is used as a tool for reasoning about or constructing computer programs written in an existing programming language. Such applications invariably require features of the programming language to be in some way mirrored in the temporal formalism. The clearest illustration of this is perhaps the compositional proof system of Barringer et al. discussed in the previous section, in which the compositional structure of programs is reflected by a similar compositionality in the temporal specifications, each programming construct being twinned with its expression in temporal logic.

To some researchers, notably Moszkowski, this twofold formalism has seemed a wasteful duplication, and accordingly attempts have been made to treat temporal logic itself as a programming language. The idea is that instead of first writing a program's specifications in temporal logic and then converting these specifications into another programming language, the program itself should be written in temporal logic. In terms of Fig. 1, this means collapsing the two upper boxes into one, so that the temporal logic formulae become programs, and their relation with the execution-sequences can be simultaneously seen both as a model-theory for the formulae qua temporal logic formulae and as an operational semantics for the formulae qua computer program. In order to do this, of course, it is necessary to construct an interpreter or compiler for temporal logic.

Moszkowski uses Interval Temporal Logic (ITL), which we discussed above (p. 42); he has adapted a subset of ITL as the programming language Tempura (Moszkowski, 1986). Moszkowski himself wrote the first Tempura interpreter in Lisp; subsequently, a faster version, written in C, has been implemented by Roger Hale. Each Tempura program is a formula of ITL.$^5$

When it is run, an interval satisfying the formula is constructed step by step, each state in the interval corresponding to an action by the computer. The uses to which Tempura can be put are described by Roger Hale in Chapter 3 of this volume.

Related work on programming in temporal logic has been done in Japan, where a temporal logic programming language called Tokio is being developed (Aoyagi et al., 1985; Fujita et al., 1986). Like Tempura, Tokio is based on ITL, but it incorporates a somewhat different subset of the full range of ITL formulae. The Tokio interpreter is implemented in Prolog, and as a result Tokio inherits such distinctive Prolog features as backtracking and unification, while at the same time allowing a more versatile handling of variables.

In Prolog, a variable can be bound to a value only once during execution of a call, so in order to perform operations like incrementing a variable by a

5. The converse is not true. This means that the specification language (ITL) is not identical to the programming language (Tempura), but rather subsumes it as a special case. The important point is that the relationship between a program and its specification can be defined formally within ITL (Roger Hale, personal communication).
fixed amount it is necessary to introduce an auxiliary variable to assume the resulting value. In Tokio, on the other hand, a variable can assume different values at different times (to be precise, a variable gets bound to a sequence of values, which can be thought of as the succession of values it assumes over an interval of time), so such operations become as simple to express as in conventional, imperative programming languages.

4 CONCLUDING REMARKS

We have seen application of ideas from temporal logic to two distinct areas of computer science: on the one hand to AI and related domains, and on the other to the theory of programs. These two areas have in fact very little to do with one another; in neither field do researchers ever have cause to refer to what has been done in the other, and the general style of work in the two areas is very different. AI work having a tendency to be discursive and at most semi-formal, while work in the theory of program verification is highly technical and mathematically rigorous. Given this, is there any justification for bringing two such disparate areas together in a single survey?

I believe that a case can be made for the desirability of greater contact between the two fields we have examined here. The very fact that the logic of time is relevant to both is suggestive of the possibility of fruitful interaction. In order to see what possible points of contact there might be, let us make a few comparisons between the two domains.

To begin with, one has to admit that although many of the same issues crop up in both areas of research, they seem to do so in ways which do not provide much common ground for discussion. For example, in both fields the issue of whether to treat time as linear or branching arises, but the criteria that have been brought to bear on this issue in the two cases are utterly different. Ben-Ari et al. (1981) stress that in program verification the decision between linear and branching time 'has very little to do with the philosophical question of the structure of physical time', but is rather 'pragmatically based on the choice of the type of programs and properties one wishes to formalise and study'. Contrast with this McDermott's decision to adopt a branching-time model, on the ground that the future really is indeterminate, and Allen's rejection of branching time on the ground that reasoning about the future is just one form of hypothetical reasoning, no different in its essentials from reasoning about the past. Despite this contrast, it remains true that both groups of researchers have to make the choice between the two approaches (and similarly with discrete versus continuous time); so it is not implausible to suggest that a systematic investigation of the formal consequences of each approach would be of assistance in either case.

Another point of contrast is that the work in reasoning about programs is at a far more advanced stage of technical sophistication. The systems of temporal logic used in that work are presented with the highest standards of mathematical rigour, and every attempt is made to provide proofs of consistency and completeness and to compute the complexity of the algorithms employed. The AI work, on the other hand, tends to be much more casual about such matters, with a plethora of axioms and definitions but little attempt to tie these together into a rigorous system which could then be investigated for consistency and completeness (similar criticisms have been made by R. Turner, 1984). It might be argued that this is an unavoidable consequence of the different nature of the tasks involved, AI work being naturally more discursive, even philosophical, in its approach. While this is certainly correct as an analysis of the cause of the difference in style, it would be lame if offered as an excuse. If a computer system that employs temporal reasoning is to be used for tasks whose successful execution is important in the real world then it is surely of the utmost importance that the temporal reasoning involved should be mathematically impeccable; otherwise one could have little confidence in the reliability of the results. So there seems to me a good case for importing into AI some of the same mathematical rigour, as regards the logical formalism, that informs the use of temporal logic in reasoning about programs.

A notable difference between the AI and program-verification approaches to time is that in the former, as we have noted, it is customary to use a first-order formalism, whereas in the latter the modal approach is well-nigh universal. AI people have tended to opt for the first-order approach because it forms the basis of existing logic programming facilities, e.g. Prolog. Perhaps if it were found possible to automate deduction in modal temporal logics with the same degree of success there would be a drift of opinion in favour of the modal approach, with its greater affinity with the temporal expressions of natural language.

Dov Gabbay's contribution to this volume (Chapter 6) represents an important step towards the realisation of this possibility. In his paper, Gabbay investigates the feasibility of extending Horn Clause logic programming by the inclusion of modal operators. The result, if successfully implemented, would be a tensed version of Prolog. Work of this kind is of immediate relevance to the problems of temporal reasoning encountered in AI, and could very likely be of assistance to software engineers as well. In any event, there is clearly much work to be done in this field, and since both the areas we are considering stand to gain by such work there is a clear case for the pooling of resources.

In conclusion, then, it may be said that although the two areas of computer science interested in temporal reasoning have had remarkably little to do with each other, there is enough common ground to allow a certain amount
of collaboration between them. In particular, recent developments in the direction of automated deduction in modal temporal logics cannot fail to be of concern to both areas, and should provide a sound basis for further interaction.

Acknowledgements

I am grateful to Roger Hale, Peter Millican, and John Tucker for reading a draft of this paper and for making numerous constructive criticisms of it.

References


2 THE USE OF TEMPORAL LOGIC IN THE COMPOSITIONAL SPECIFICATION OF CONCURRENT SYSTEMS

Howard Barringer

Department of Computer Science, University of Manchester

ABSTRACT

This chapter considers the application of temporal logics in the formal specification and development of complex computing systems. In particular, the relevance of compositional proof theories, modularity and abstractness are motivated. A basic technique for obtaining compositional program proof theories, using a temporal language as the assertion language, is demonstrated. Several specialisations for various real parallel programming language frameworks are indicated. Finally, a problem in obtaining abstract semantic descriptions of programming languages in the temporal framework is discussed, together with one particular solution suggesting the use of a logic based on a linear dense time model.

1 INTRODUCTION

Over the last 10–15 years, rapid advances in technology have brought computing devices in one form or another to the everyday life of the general public. We find micro-computer devices in such diverse systems as:

- pocket calculators, watches, washing machines, microwave ovens, telephones, electronic games, etc.
- banking systems, automated office environments, etc.
- patient-monitoring systems, etc.
- flight control systems, nuclear reactor control systems, etc.
- defence (and attack) systems.


