Formal Aspects of Computing (1995) 7: 533–549
© 1995 BCS

Formal Aspects of Computing

METATEM: An Introduction

H. Barringer[1], M. Fisher[2], D. Gabbay[3], G. Gough[1] and R. Owens[4]

[1] Department of Computer Science, University of Manchester, Manchester
[2] Department of Computing, Manchester Metropolitan University, Manchester
[3] Department of Computing, Imperial College of Science, Technology and Medicine, London
[4] Nomura Research Institute Europe Ltd., London

Keywords: Modal and temporal logics; Reactive systems; Specification; Prototyping; Mechanical verification; Non-procedural languages; Logic programming

Abstract. In this paper a methodology for the use of temporal logic as an executable imperative language is introduced. The approach, which provides a concrete framework, called METATEM, for executing temporal formulae, is motivated and illustrated through examples. In addition, this introduction provides references to further, more detailed, work relating to the METATEM approach to executable logics.

1. Introduction

The purpose of this paper is to introduce and motivate the methodology of temporal logic as an executable imperative language, originally presented by Moszkowski [Mos86] and Gabbay [Gab87a], and to present a framework, called METATEM [BFG89a], for executing temporal logics. This methodology serves as the natural meeting ground for the declarative and imperative approaches in computing, namely *imperative logic*.

This article is a shortened version of a full paper, which is available by ftp [BFG95]. The presentation here provides a starting point for exploration of more detailed work on the METATEM family of languages.

Correspondence and offprint requests to: Michael Fisher, Department of Computing, Manchester Metropolitan University, Chester Street, Manchester M1 5GD, UK. M.Fisher@doc.mmu.ac.uk

2. Motivation

We distinguish several alternative readings or views of logic: declarative, procedural and imperative. The declarative view is the traditional one, manifesting itself both syntactically and semantically. Syntactically, a logical system is taken as being characterised by its set of theorems. It is unimportant how these theorems are generated, indeed, two different algorithmic systems generating the same set of theorems are considered as producing the same logic. Semantically, a logic is considered as a set of formulae which are satisfied by all models. A model, $\mathcal{M}$, is a static semantic object. We evaluate a formula φ in a model and, if the result of the evaluation is positive (notation $\mathcal{M} \models \varphi$), the formula holds. Thus the logic obtained is the set of all formulae characterising some particular class of models.

Applications of logic in computer science have mainly concentrated on the exploitation of its declarative features, i.e., where logic is taken as a language for describing properties of models. This view of logic is, for example, most suitably and successfully exploited in the areas of databases, program specification and program verification. A database can be presented as a deductive logical theory which is queried using logical formulae. The process of logical evaluation corresponds to the computational querying in the database. In program verification, for example, the semantics of the programs being studied can be described as logical formulae. The description plays the role of a model $\mathcal{M}$. A specification, or desired property, is then given as a logical formula φ, and the query whether φ holds in $\mathcal{M}$ (denoted $\mathcal{M} \vdash \varphi$) amounts to verifying that the program satisfies the specification. These methodologies rely solely on the declarative nature of logic.

Logic programming as a discipline is essentially declarative; in fact, it advertises itself as such. It is most successful in areas where the declarative component is dominant, for example deductive databases, though its procedural features are computational. In the course of evaluating whether $\mathcal{M} \vdash \varphi$, a *procedural reading* of $\mathcal{M}$ and φ is used. The formula φ does not act imperatively on $\mathcal{M}$, the declarative logical features are used to guide a procedure, that of taking steps for finding whether φ is true. What does not happen as part of this process is any imperative reading of φ, resulting in some action. In logic programming, such actions (e.g. assert) are obtained as side-effects through special non-logical imperative predicates and are considered undesirable. There is certainly no conceptual framework within logic programming for properly identifying only those actions which have logical meaning.

The declarative view of logic allows for a variety of logical systems. Given a set of data Δ and a formula φ, we can denote by $\Delta\, ?\, \varphi$ the query of φ from Δ. Different procedures or logics will give different answers, e.g. $\Delta \vdash_{L1} \varphi$ or $\Delta \nvdash_{L2} \varphi$ depending on the logic. Temporal and modal logics allow for systems of databases as data, so the basic query/data situation becomes

$$\{t_i : \Delta_i\}\ ?\, t : \varphi$$

meaning "if Δ_i hold at t_i then we query whether φ holds at t?".

The t_i may be related somehow, for example, they may represent moments in time. The basic situation is still the same. We have (distributed) data and we ask a specific query. The different logical systems (modal, temporal) have to do with the representation and reasoning with the distributed data. There are various approaches. Some use specialised connectives to describe the relations between

$t_i : \Delta_i$. Some use a metalogic to describe it, for example classical logic. Some reason directly on labelled data. They all share the common theme that they answer queries from given databases, however complex they are and whatever the procedure for finding the answer is.

Implicit in the data/query approach is an external view of the system. We look from the outside at data, at a logic and at a query, and we can check from the outside whether the query follows from the data in that logic. Temporal data can also be viewed this way. However, in the temporal case, we may not necessarily be external to the system. We may be within it, moving along with the temporal flow. When we view the past, we can be outside, collect the data and query it in whatever logic we choose. This is the declarative approach. When we view the future, from where we are, we have an option. We can either wait for events to happen and query them, or we can influence events as they occur, or even make them occur if we can. The result of this realisation is the imperative view of the future. A temporal statement like "John will wait for two hours" can only be read declaratively as something evaluated at time t by an observer at a later time (more than two hours). However, an observer at time t itself has the further option of reading it imperatively, resulting in the action of forcing John to wait for two hours. In effect what we are doing is dynamic model construction. To summarise, we distinguish three readings of logical formulae: the traditional declarative reading, finding answers from data; the logic programming procedural reading, supporting the declarative and giving procedures for finding the answer; the imperative reading which involves direct actions.

Reading the future imperatively in this manner is the basis of our imperative use of temporal logic as a paradigm for a new executable logic language. It is the theme this paper introduces — the idea that a future statement can be read as commands. In traditional declarative uses of logic there are some aspects of this idea. For example, in logic programming and deductive databases the handling of integrity constraints borders on the use of logic imperatively. Integrity constraints have to be maintained. Thus one can either reject an update or carry out some corrections to the database. Maintaining integrity constraints is a form of executing logic, but it is logically *ad hoc* and has to do with the local problem at hand. Truth maintenance is another form. In fact, under a suitable interpretation, one may view any conflict resolution mechanism as model building, which can in turn be seen as a form of execution. In temporal logic, model construction can be interpreted as execution. Generating the model, i.e. finding the truth values of the atomic predicates in the various moments of time, can be taken as a sequence of execution.

As the need for the imperative executable features of logic is widespread in computer science, it is not surprising that various researchers have touched upon it in the course of their work. In spite of this, there has been no conceptual methodological recognition of the imperative paradigm in the community, nor has there been a systematic attempt to develop and bring this paradigm forward as a new and powerful logical approach in computing.

The logic USF, defined by Gabbay [Gab87a], was a first attempt at promoting the imperative view as a methodology. In this paper, we consider a more refined language based on this imperative view of temporal logic.

3. Executing the Future — The METATEM Approach

Consider a temporal sentence of the form:

antecedent (about the past) $\Rightarrow$ consequent (about the present and future)

This can be interpreted as **if** the "antecedent (about the past)" is true **then do** the "consequent (about the present and future)". Adopting this imperative reading yields us an execution mechanism for temporal logics. We take this as the basis of our approach, called METATEM. This name, in fact, captures two key aspects of our developing framework:

- the use of TEMporal logic as a vehicle for specification and modelling, via direct execution of the logic, of reactive systems;
- the direct embodiment of META-level reasoning and execution, in particular, identifying metalanguage and language as one and the same, providing a highly reflective system.

Generally, the behaviour of a reactive component is specified by describing the interactions that occur between that component and the context, or environment, in which it is placed. In particular, a distinction needs to be made between actions made by the component and those made by the environment. In METATEM, we enforce such a distinction. The behaviour of a component is described by a collection of temporal rules, in the form mentioned above. The occurrence, or otherwise, of an action is denoted by the truth, or falsity, of a proposition. The mechanism by which propositions are linked to actual action occurrences is not an issue here and is left to the reader's intuition. However, the temporal logic used below does distinguish between component and environment propositions.

A METATEM program for controlling a process is presented as a collection of temporal rules. The rules apply universally in time and determine how the process progresses from one moment to the next. A temporal rule is given in the following clausal form

(past time antecedent) $\Rightarrow$ (present and future time consequent)

This is not an unnatural rule form and occurs in some guise in many programming languages. For example, in imperative programming languages it corresponds to the conditional statement. In declarative logic programming, we have the Horn clause rule form of Prolog and other similar languages. In METATEM, the "past time antecedent" is a temporal formula referring strictly to the past; the "present and future time consequent" is a temporal formula referring to the present and future. Although we can adopt a declarative interpretation of the rules, for programming and execution purposes we take an imperative reading following a natural view of the way in which dynamic systems behave and operate, namely,

on the basis of the past **do** the present and future.

Let us first illustrate this with a simple merging problem. There are two queues, let us say at Heathrow Airport, queue 1 for UK nationals, and queue 2 for other nationalities. The merge point is the immigration desk. There is a man in charge, calling passengers from each queue to go through immigration. This is a merge problem. We take "snapshots" each time a person moves through. The propositions we are interested in are, "from which queue we are merging" and "whether the UK nationals' queue is longer than the other queue". We must not upset the first queue (UK nationals); these people vote, the others don't. We of course generate a time model, as we proceed with the merging.

We want to specify that if queue 1 is longer, we should next merge from queue 1. The desired specification is written in temporal logic. Let *m1* be "merge from queue 1", *m2* be "merge from queue 2" and *b* be "queue 1 is longer than queue 2". Note that *m1* and *m2* are controlled by the program, and *b* is controlled by the environment. The specification is the conjunction of the three formulae:

$$\Box(m1 \vee m2)$$
$$\Box\neg(m1 \wedge m2)$$
$$\Box(b \Rightarrow \bigcirc m1).$$

Here '$\Box$' denotes the "always in the future" operator and '$\bigcirc$' denotes the "next moment in time" operator.

We actually want to use this *specification* to control the merge while it is happening. To explain our idea look at the specification $b \Rightarrow \bigcirc m1$. If we are communicating with the merge process and at time n we see that b is true, i.e. queue 1 is longer than queue 2, we know that for the specification to be true, $\bigcirc m1$ must be true, i.e. $m1$ must be true at time $n+1$. Since we are at time n and time $n+1$ has not come yet, we can go ahead and make $m1$ true at time $n+1$. In other words, we treat $\bigcirc m1$ as the imperative statement, "execute *m1* next".

The specification, $b \Rightarrow \bigcirc m1$, which is a declarative statement of temporal logic, is thus being perceived as an executable imperative statement of the form:

If *b* **then** next **do** *m1*.

In practice we simply tell the merge process to merge from queue 1. Our point of view explains the words "the imperative future". Of course when adopting this point of view, we must know what it means to **do** or **execute** *m1* or indeed any other atom or formula. To **execute** *mi* we merge from *queuei*. How do we **execute** *b*? We cannot make the queue longer under the conditions of the example at Heathrow. In this case we just helplessly wait to see if more passengers come.

More specifically, given a specification expressed by a formula φ in a connective-based temporal logic, we can use the separation theorem for discrete temporal logics (we assume we have a fully expressive set of connectives) to rewrite the specification φ in the form

$$\Box \bigwedge_i (\varphi^i \Rightarrow \psi_1^i \vee \psi_2^i)$$

where φ^i is a pure past formula, ψ_1^i is a boolean expression of atoms and ψ_2^i is a pure future formula [Gab87a].

As we have seen above, there may be various ways of separating the original formula φ. The exact rewrite formulation requires an understanding of the application area. This is where the ingenuity of the programmer is needed. Having put φ in separated form we read each conjunct

$$\varphi^i \Rightarrow \psi_1^i \vee \psi_2^i$$

as:

If φ^i is true at time n **then do** $(\psi_1^i \vee \psi_2^i)$

Proposition Symbols

$a, b \ldots \in \mathcal{A}_E$

$x, y \ldots \in \mathcal{A}_C$

Note: $\mathcal{A}_P = \mathcal{A}_E \cup \mathcal{A}_C$ and $\mathcal{A}_E \cap \mathcal{A}_C = \emptyset$

Classical Propositional Connectives

true **false**

$\neg \quad \wedge \quad \vee \quad \Rightarrow \quad \Leftrightarrow$

Temporal Connectives

unary	$\bigcirc$	next
	$\bullet$	last
	$\square$	always in future
	$\blacksquare$	always in past
	$\diamond$	sometime in future
	$\blacklozenge$	sometime in past
binary	$\mathcal{W}$	unless
	$\mathcal{Z}$	zince
	$\mathcal{U}$	until
	$\mathcal{S}$	since

Fig. 1. Basic symbols.

More formally

$$\textbf{Hold}\ (\varphi^i) \Rightarrow \textbf{Execute}\ (\psi_1^i \vee \psi_2^i)$$

We must make it clear though that there is a long way to go if want to build a practical system. In the following sections, we introduce our base propositional temporal logic, outline a general interpretation strategy and provide an example execution.

4. A Propositional Linear-time Temporal Logic

In order to give a flavour of the METATEM execution style, we present a simple propositional temporal logic and outline the execution mechanism for this restricted logic. Although not as powerful as it might be, such a logic does provide a basis for discussion of many of the novel aspects of METATEM.

We introduce a basic *linear* and *discrete* propositional temporal logic, PML (Propositional METATEM Logic). The language is obtained by augmenting classical propositional logic with *temporal* operators. The reader who is familiar with such standard temporal logics may prefer to skip the next two subsections and proceed to Section 5 where our execution approach is outlined.

The logic is presented in the usual way: the next sub-section introduces the syntactic elements, while the following sub-section introduces the semantics.

4.1. Syntax

The basic symbols of the logic are depicted in Fig. 1. We assume an alphabet of proposition symbols $\mathcal{A}_P$ which is the union of two disjoint sets $\mathcal{A}_C$ and $\mathcal{A}_E$ of propositions controlled by the component and by the environment, respectively. A standard collection of propositional connectives together with a collection of temporal connectives complete the basic symbol: $\bigcirc$, $\square$ and $\diamondsuit$ are unary *future*-time connectives; $\mathcal{W}$ and $\mathcal{U}$ are binary *future*-time connectives; $\bullet$, $\blacksquare$ and $\blacklozenge$ are unary *past*-time connectives and $\mathcal{Z}$ and $\mathcal{S}$ are binary *past*-time connectives.

The formulae of this propositional temporal logic are constructed inductively in the normal way. Thus, propositions p from $\mathcal{A}_P$ are formulae and, given formulae φ and ψ, so are the propositional combinations **true**, **false**, $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \Rightarrow \psi$ and $\varphi \Leftrightarrow \psi$, and the temporal combinations, $\bigcirc\varphi$, $\square\varphi$, $\diamondsuit\varphi$, $\bullet\varphi$, $\blacksquare\varphi$, $\blacklozenge\varphi$, $\varphi\mathcal{W}\psi$, $\varphi\mathcal{U}\psi$, $\varphi\mathcal{Z}\psi$ and $\varphi\mathcal{S}\psi$. The binding of the propositional connectives is assumed prioritised in the order given, i.e. (highest) $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ (lowest). The unary temporal connectives, i.e. $\bigcirc$, $\square$ and $\diamondsuit$, have binding equal in priority to negation, i.e. $\neg$. The binary temporal connectives have priority in between $\neg$ and $\wedge$. However, as usual, to change the effect of binding, bracketing of formulae is allowed; so, given a formula φ, then (φ) is also a formula.

We refer to the syntactic class of well-formed formulae defined informally above as *Wff*. It is also useful to define the subclasses $Wff_<$, $Wff_=$, $Wff_\geq$, covering, respectively, strict past time formulae, present time formula and non-strict future time formulae.

$Wff_=$ is the set which includes the propositions p from $\mathcal{A}_P$ and is closed under the propositional connectives.

$Wff_<$ is the set which includes the formulae $\bullet\phi$, $\blacksquare\phi$, $\blacklozenge\phi$, $\phi\mathcal{S}\psi$ and $\phi\mathcal{Z}\psi$, where ϕ and ψ are in $Wff_\leq$, and is closed under unary and binary propositional connectives. $Wff_\leq$ includes the set of propositions and is closed under the propositional and past-time temporal connectives.

$Wff_\geq$ is the set which includes $Wff_=$ and is closed under the propositional and future-time temporal connectives.

4.2. Semantics

We interpret temporal formulae over linear, discrete structures, indeed, one might refer to the temporal flow as "natural number time". More formally, models are taken as sequences $\sigma \in (\mathbf{N} \rightarrow 2^{\mathcal{A}_P})$, and formulae are interpreted in structures (σ, i), $i \in \mathbf{N}$. Thus, σ provides an interpretation for the atomic propositions of the language at different points in time. Given some moment in time, represented by $j \in \mathbf{N}$, $\sigma(j)$ is a set of propositions drawn from the alphabet $\mathcal{A}_P$ and denotes all those propositions that are to be taken as true at that moment j. σ (representing a sequence) is total with respect to $\mathbf{N}$. The i component of the pair (σ, i) is an index which denotes the current moment, i.e. *now*, in the model (referred to as the *reference point* of the model). Indices greater than i represent future moments and indices less than i represent past moments.

The temporal model is *linear* as for every moment j there is exactly one future given by the points $j+1$, $j+2$, etc. The model is *discrete* as it is based on the natural numbers. Furthermore, as the mapping is assumed total over the naturals, time is infinite in the future and finite, or bounded, in the past. Due of the linearity and discreteness properties, the σ mapping can be conveniently thought a sequence of *states*, e.g. $s_0, s_1, s_2, s_3, \ldots$.

A relation $\models$ is defined inductively over the structure of formulae and provides an interpretation for temporal formulae in the given model structures. Figure 2

Propositions

$$(\sigma,i) \models p \quad \text{iff} \quad p \in \sigma(i)$$

Propositional Connectives

$$\begin{array}{lll}
(\sigma,i) \models \mathbf{true} & & \\
(\sigma,i) \models \neg\varphi & \text{iff} & \text{not } (\sigma,i) \models \varphi \\
(\sigma,i) \models \varphi \wedge \psi & \text{iff} & (\sigma,i) \models \varphi \text{ and } (\sigma,i) \models \psi \\
\text{etc.} & &
\end{array}$$

Temporal Connectives

$$\begin{array}{lll}
(\sigma,i) \models \bigcirc\varphi & \text{iff} & (\sigma,i+1) \models \varphi \\
(\sigma,i) \models \varphi\,\mathcal{U}\psi & \text{iff} & \text{for } \mathbf{some}\ k \in \mathbb{N}.\ \ (\sigma,i+k) \models \psi \text{ and} \\
 & & \quad\text{for } \mathbf{all}\ j \in 0..k-1,\ \ (\sigma,i+j) \models \varphi \\
(\sigma,i) \models \bullet\varphi & \text{iff} & i = 0 \text{ or } (\sigma,i-1) \models \varphi \\
(\sigma,i) \models \varphi\,\mathcal{S}\,\psi & \text{iff} & \text{for } \mathbf{some}\ k \in 1..i\ \ (\sigma,i-k) \models \psi \text{ and} \\
 & & \quad\text{for } \mathbf{all}\ j \in 1..k-1,\ (\sigma,i-j) \models \varphi
\end{array}$$

Fig. 2. Interpretation in a model structure.

$$\begin{array}{lllllll}
\Diamond\varphi & \Leftrightarrow & \mathbf{true}\,\mathcal{U}\varphi & \qquad & \blacklozenge\varphi & \Leftrightarrow & \mathbf{true}\,\mathcal{S}\,\varphi \\
\Box\varphi & \Leftrightarrow & \neg\Diamond\neg\varphi & & \blacksquare\varphi & \Leftrightarrow & \neg\blacklozenge\neg\varphi \\
\varphi\mathcal{W}\psi & \Leftrightarrow & \varphi\,\mathcal{U}\psi \vee \Box\varphi & & \varphi\,\mathcal{Z}\,\psi & \Leftrightarrow & \varphi\,\mathcal{S}\,\psi \vee \blacksquare\varphi
\end{array}$$

Fig. 3. Other temporal definitions.

provides its definition. Of particular interest is the interpretation given to a strong until formula. $\varphi\,\mathcal{U}\psi$ is true in model (σ,i) if and only if (i) ψ is true eventually in that model, say with reference point $i+k$ $(k \geq 0)$, and (ii) the formula φ is true for all the models $(\sigma,i+j)$ where the reference points $i+j$ ranges up to but not including the reference point for which ψ is true. Note that the formula $\varphi\,\mathcal{U}\psi$ is true at i in any model for which ψ is true at i and therefore $\psi \Rightarrow \varphi\,\mathcal{U}\psi$ is true for every model.

The past time temporal operators are defined to be *strict* past time versions of their future time counterparts, i.e. the past does not include the present moment. Since the model structure allows easy backwards as well as forwards reasoning the interpretations given should be self evident. However, it is worth discussing the interpretation of $\bullet$ at the beginning of time. The last-time operator, $\bullet$, has been given a weak interpretation, i.e. for any formula φ, $\bullet\varphi$ is true at the beginning of time. Hence the formula $\bullet\mathbf{false}$ can be used to determine the beginning of time, i.e. it is true only at the beginning of time. One can define a strong (existential) last time operator, denoted here by $\odot$, such that $\odot\varphi$ is false at the beginning of time for any φ. Then, of course, notice the duality between the two last time operators, i.e. $\odot\varphi \Leftrightarrow \neg\bullet\neg\varphi$.

The interpretation of formulae constructed from the other temporal operators, i.e. $\Box$, $\Diamond$, $\mathcal{W}$, $\blacksquare$, $\blacklozenge$, $\mathcal{Z}$, is given by definition in terms of $\mathcal{U}$ and $\mathcal{S}$, see Fig. 3.

We now state some properties of PML.

Definition 4.1. A formula φ is said to be *satisfied* by a model σ at i if, and only if, $(\sigma, i) \models \varphi$.

Definition 4.2. A formula φ is said to be *validated* by a model σ if, and only if, $(\sigma, i) \models \varphi$ holds for every index i.

The definition of validity is as in classical logic.

Definition 4.3. A formula φ is said to be *valid*, $\models \varphi$, if, and only if, it is validated by all models σ.

Theorem 4.1. Validity in PML is decidable.

Proof. See [Gou84], for example. $\Box$

This decidability result is important. It means that algorithms can be constructed to determine the validity, or otherwise, of any given PML formula. Thus, for programs whose semantics can be represented by a propositional temporal formula, this result provides an automatic verification method for temporal logic specifications (see, for example, [LPZ85, BFG89b]). As we shall see, the decidability of PML is also of importance for "execution".

The following Separation Theorem, due to Gabbay in 1980, provides an initial foundation for the METATEM approach.

Theorem 4.2. For any formula φ in *Wff*, we can find a *separated* formula, φ', consisting of a boolean combination of strict past time formulae (in $Wff_<$), present time formulae (in $Wff_=$) and strict future time formulae (in $Wff_>$), such that φ and φ' are semantically equivalent, i.e. every model of φ is a model for φ', and *vice versa.*

Proof. See [Gab87a], for example. $\Box$

From the above result, it follows that:

Theorem 4.3. Any formulae φ of class *Wff* can be written in the form

$$\Box \bigwedge_{i=1}^{n} (\xi_i \Rightarrow \psi_i)$$

where ξ_i are strict past time formulae, i.e. in $Wff_<$, and ψ_i are non-strict future time formulae, i.e. in $Wff_\geq$, such that both the above formula and φ satisfy the same set of models at index (time point) 0 (or, indeed, any particular index k).

Proof. Suppose φ is to be evaluated at time point k. First separate φ into a boolean combination of strict past, present and strict future formulae. By the Separation Result, this boolean combination will be satisfy the same set of models at time point n. Put the boolean combination into the form

$$\bigwedge_{i=1}^{n} (\phi_i \Rightarrow \psi_i)$$

where ϕ_i is in $Wff_<$ and each ψ_i is in $Wff_\geq$. To obtain the desired form

$$\Box \bigwedge_{i=1}^{n} (\xi_i \Rightarrow \psi_i)$$

construct each ξ_i formula as $\bullet^k\mathbf{false} \wedge \phi_i$. $\Box$

Finally note that:

Theorem 4.4. Given a formula ψ in $Wff_{\geq}$, it can be written in a logically equivalent form as

$$\bigvee_{i=1}^{n} f_i$$

where each f_i is either of the form $\varphi_i \wedge \bigcirc\psi_i$ or of the form φ_i where each φ_i is a conjunction of literals, i.e. a present time formula from $Wff_{=}$, and ψ_i is a future (non-strict) time formula, i.e. in $Wff_{\geq}$.

5. METATEM Programs and Their Execution

As outlined above, a METATEM program for controlling a reactive component is presented as a collection of temporal rules. which apply universally in time and determine how the process progresses from one moment to the next. A temporal rule is given in the following clausal form

(past time antecedent) $\Rightarrow$ (present and future time consequent).

In METATEM, the "past time antecedent" is a temporal formula referring strictly to the past, i.e. it is in $Wff_{<}$; the "present and future time consequent" is a temporal formula referring to the present and future, i.e. it is in $Wff_{\geq}$. Although we can adopt a declarative interpretation of the rules, for programming and execution purposes we take an imperative reading following the natural way we ourselves tend to behave and operate, namely,

on the basis of the past **do** the present and future.

Given a program consisting of a set of rules R_i, this imperative reading results in the construction of a model for the formula

$$\Box \bigwedge_{i} R_i$$

which we refer to as the *program formula.* The execution of a METATEM program proceeds, informally, in the following manner. Given some initial history of execution:

1. Determine which rules currently apply, i.e. find those rules whose past time antecedents evaluate to true in the current history;
2. "Jointly execute" the consequents of the applicable rules together with any commitments carried forward from previous times — this will result in the current state being completed and the construction of a set of commitments to be carried into the future;
3. Repeat the execution process for the next moment in the context of the new commitments and the new history resulting from (2) above.

$$\bigcirc r_1 \Rightarrow \Diamond a_1 \qquad (1)$$
$$\bigcirc r_2 \Rightarrow \Diamond a_2 \qquad (2)$$
$$(\neg r_1 \ \mathcal{Z}\ (a_1 \wedge \neg r_1)) \Rightarrow \neg a_1 \qquad (3)$$
$$(\neg r_2 \ \mathcal{Z}\ (a_2 \wedge \neg r_2)) \Rightarrow \neg a_2 \qquad (4)$$
$$\bullet \mathbf{true} \Rightarrow (\neg a_1 \vee \neg a_2) \qquad (5)$$

Fig. 4. Temporal rules for resource manager.

Requests		*Allocations*		*Time*	*Commitments*
r_1	r_2	a_1	a_2	*Step*	
y	n	n	n	0	
n	n	y	n	1	
y	y	n	n	2	
n	n	y	n	3	$\Diamond a_2$
y	n	n	y	4	
n	y	y	n	5	
n	n	n	y	6	

Fig. 5. Sample execution of resource manager.

6. Example

To demonstrate the execution process we offer the following simple example. Consider a resource being shared between several (distributed) processes, for example a database lock. We require a "resource manager" that satisfies the following constraints.

1. If the resource is requested by a process then it must eventually be allocated to that process.
2. If the resource is not requested then it should not be allocated.
3. At any one time, the resource should be allocated to at most one process.

To simplify the exposition of the execution process, we restrict the example to just two processes. Let us use propositions r_1 and r_2 to name the occurrence of a request for the resource from process 1 and process 2 respectively. Similarly, let propositions a_1 and a_2 name appropriate resource allocations. The request propositions, r_i, are controlled by the environment of the resource manager, whereas the allocation propositions are under direct control of the resource manager and can not be modified by the environment. Writing the given informal specification in the desired rule form, i.e. "pure past formula implies present and future formula", results in the rules of Fig. 4.
The state provides an interpretation for the four propositions r_1, r_2, a_1 and a_2. Given particular settings for the environment propositions, i.e. r_1 and r_2, the execution will determine appropriate values for the allocation propositions, a_1 and a_2. Figure 5 gives the first seven steps of a typical trace.

Step 0. The environment has requested the resource for process 1. To see how the current state is completed, we must find which rules from Fig. 4 apply. Clearly

rules 1 and 2 do not apply; we are currently at the beginning of time and hence $\bullet\phi$, for any ϕ, is false. The other rules do apply and require that both a_1 and a_2 are made false. No commitments are carried forward. Execution proceeds to the next time step.
Step 1. First, there are no new requests from the environment. However, in examining which rules apply, we note that the hypothesis of rule 1 is true (there was a request in the previous moment), hence we must "execute" $\Diamond a_1$. Also, rule 4 and rule 5 apply. In fact, the latter rule applies at every step because its hypothesis is always true. To execute $\Diamond a_1$, we execute $a_1 \vee (\neg a_1 \wedge \bigcirc\Diamond a_1)$. We have a choice; however, our mechanism will prioritise such disjunctions and attempt to execute them in a left to right order. One reason for this is that to satisfy an eventuality such as $\Diamond\varphi$, we must eventually, in some future time step, satisfy φ; so we try to satisfy it immediately. If we fail then we must carry forward the commitment to satisfy $\Diamond\varphi$. Here we can make a_1 true. Rule 4 requires that a_2 is false, leaving rule 5 satisfied.
Step 2. The environment makes a request for both process 1 and process 2. The hypotheses of rules 1 and 2 are false, whereas those of rules 3 and 4 are true. So a_1 and a_2 are made false. No commitments are carried forward.
Step 3. There are no new requests from the environment, but there are two outstanding requests, i.e. both rule 1 and 2 apply. However, rule 5 requires that only one allocation may be made at any one time. The execution mechanism "chose" to postpone allocation to process 2. Thus, the eventuality from rule 1 is satisfied, but the eventuality for rule 2 is postponed, shown in the figure by carrying a commitment $\Diamond a_2$.
Step 4. A further request from the environment occurs, however, of interest here is the fact that $\Diamond a_2$ must be satisfied in addition to the commitments from the applicable rules. Note that this time, rule 4 does not apply as there is an outstanding request. Fortunately, the execution mechanism has no need to further postpone allocation to process 2 and a_2 is made true.
Steps 5 – 6. Similar to before.

7. Related Work

The study of temporal and modal logics in the formal development of reactive systems has, in the main, been concentrated on techniques for specification and verification. On a related theme, there have been investigations into the synthesis of programs from temporal logic based specifications, for example [MaW84] and more recently [PnR89a, PnR89b].

The direct execution of temporal logic itself has been studied by a variety of researchers over the past decade. These systems can be categorised in many ways, for example by their underlying execution paradigm (logic programming or imperative logic) or by the type of temporal logic executed (discrete linear, branching or interval-based).

7.1. Executing Discrete Temporal Logics

The main alternative to the imperative reading of discrete temporal logics described in this paper, is to execute such logics using the logic programming paradigm. A temporal logic programming language, called TEMPLOG, was initially defined by Abadi and Manna [AbM89] and has subsequently been studied in order to establish it's expressive power and completeness properties [Bau89, Bau92]. A related approach is based upon the temporal extension of intensional logic programming, and is called CHRONOLOG [OrW92b, OrW92a].

Temporal Prolog [Gab91] provides an alternative (to TEMPLOG) extension of the logic programming paradigm to discrete temporal logics.

Brzoska [Brz95] presents an extension to temporal logic programming of the TEMPLOG form, incorporating not only past-time operators, but also metric temporal operators. He gives the correctness of the logic programming system in his framework and shows how it can be translated into a constraint logic programming system over an appropriate algebra.

Merz [Mer95] discusses the issues involved in the tradeoff between efficiency and expressiveness in the execution of temporal formulae. In particular, he defines a class of temporal logic formulae, which is both sufficiently abstract to support high-level descriptions of algorithms and, yet, whose model construction problem is tractable.

Tang [Tan89], rather than providing an extension of Prolog that incorporates a form of temporal logic, extends the verification method for temporal logic to incorporate logic programming. In particular, the states of a model checker for CTL (a branching-time temporal logic [EmC82]) are extended with Prolog like statements.

7.2. Interval Temporal Logics

Two different varieties of interval temporal logics form the basis for programming languages. These are ITL, developed by Moszkowski for the purpose of modelling digital circuits [Mos83], and Allen's Interval Algebra, developed in order to provide a formal foundation for temporal representation and temporal planning, particularly in AI [All84].

Tempura [Mos86, HaM87, Hal87] is an executable temporal logic based upon the forward chaining execution of ITL. In this sense it was the precursor of the METATEM family of languages and provides a simpler and more tractable alternative to these approaches.

Tokio [FKT86] is a logic programming language based on the extension of Prolog with ITL formulae. It provides a powerful system in which a range of applications can be implemented and verified.

Hrycej [Hry88, Hry93] describes a completely different *Temporal Prolog* from Gabbay's (see above), which is based upon interval temporal logic. He extends Prolog using a form of Allen's interval temporal logic [All84] and applies it to temporal knowledge representation and temporal planning problems. In addition to the basic Prolog execution mechanism, such a system requires a constraint solver for temporal constraints (not unlike constraint logic programming [JaL87]), though this itself might be implemented in Prolog.

Frühwirth [Frü95] outlines another method for implementing interval temporal logics as an extension to logic programming, this time by defining a *temporal annotated logic*. This can be mapped directly on to a particular form of constraint logic programming.

Introductory surveys of executable temporal logics can be found in [OrM94] and [FiO95a], while a more detailed account of some of the current research in this area can be found in the proceedings of the the first international workshop on executable modal and temporal logics [FiO95b].

8. Further Reading

While this paper has provided an introduction to the MetateM approach, there are many details and extensions that have not been covered here. In this section, we reference, and briefly describe, some of this, more detailed, work.

Applications

MetateM has been applied in a variety of areas such as reactive system simulation, temporal databases, and the modeling of large 'real-life' deductive systems [FFO93, FMO91, ToM90]. For example, in [FFO93], MetateM rules are used to simulate a simple railway network.

Implementation

Initial interpreters for MetateM were developed by both Michael Fisher and Richard Owens; this work is reported in more detail in [FiO92], and incorporates the execution of temporal formulae, but based upon a normal form, called SNF [Fis92], rather than arbitrary formulae. The use of this normal form both simplifies the execution mechanism and obviates the need for a separation theorem for the logic (as renaming can be used to transform arbitrary formulae into SNF).

If propositional MetateM programs contain no references to environment variables then the execution mechanism can be viewed as tracing out a subtree of a semantic tableau associated with the program rules. The tree produced is effectively a graph representing the basic automaton for the formula being executed. Thus, MetateM programs can be 'compiled' into automata in this way. In this case, the execution mechanism would proceed by exploring paths through the graph in an attempt to follow an infinite branch satisfying the eventualities. Although unsatisfiable branches may be followed for a finite number of states, the invocation of the loop-checking mechanism will force the execution mechanism to backtrack from such branches and explore different branches. Eventually, the execution mechanism will follow an infinite satisfying branch. Future implementations of propositional (non-concurrent) MetateM will be based upon this "compilation to automata" approach.

Concurrency

Although our work on MetateM had shown its utility in the development of a variety of sequential deductive systems, the need for an extended version of MetateM exhibiting concurrency was apparent. Thus, Concurrent MetateM was developed, providing an operational model for concurrently executing, independent MetateM processes [BG88, FiB91, Fis93, Fis94]. These independent processes execute their own MetateM specifications, and are able to communicate with each other through asynchronous broadcast message-passing. The processes themselves are completely autonomous, having control not only over their own execution, but also over their interaction with the environment.

Further, individual processes are able to dynamically change their internal behaviour and can choose to participate in group actions. Not surprisingly, Concurrent METATEM has been shown to be useful in a variety of applications, particularly those involving reactive systems and Distributed Artificial Intelligence [FFO93, Fis93, FiW93].

Concurrent METATEM is not a direct extension of METATEM but, although it both restricts the general METATEM model and extends the potential for concurrency, it is still based upon the *imperative* reading of temporal logic.

Meta-Programming

A further aspect of METATEM is its potential for meta-programming. An initial investigation into this capability is provided in [BFG91].

Synthesis

One of the main performance problems with METATEM programs is in applications where a large amount of backtracking occurs. Thus, just as in logic programming where synthesis procedures have been developed in order to reduce non-determinism, semi-automatic temporal synthesis procedures have been investigated for METATEM programs [Noë91, FiN92]. Initial results for propositional METATEM programs are promising and further work is planned towards the reduction of non-determinism in full first-order METATEM.

9. Summary

In this paper, we have introduced the foundations of an *imperative* approach to the execution of temporal logics. We have indicated, not only how dynamic properties of systems can be represented within METATEM programs, but also how such programs may be implemented. Finally, we have also provided references to the wide range of work, both completed and in progress, on the METATEM family of languages.

References

[All84] Allen, J.: Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[AbM89] Abadi, M. and Manna, Z.: Temporal Logic Programming. *Journal of Symbolic Computation*, 8: 277–295, 1989.

[Bau89] Baudinet, M.: Temporal Logic Programming is Complete and Expressive. In *Proceedings of Sixteenth ACM Symposium on Principles of Programming Languages (POPL)*, Austin, Texas, January 1989. ACM.

[Bau92] Baudinet, M.: A Simple Proof of the Completeness of Temporal Logic Programming. In L Fariñas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*. Oxford University Press, 1992.

[BFG89a] Barringer, H., Fisher, M., Gabbay, D., Gough, G. and Owens, R.: METATEM: A Framework for Programming in Temporal Logic. In *Proceedings of REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, Mook, Netherlands, June 1989. (Published in *Lecture Notes in Computer Science*, volume 430, Springer Verlag).

[BFG95]* Barringer, H., Fisher, M., Gabbay, D., Gough, G. and Owens, R.: METATEM: An Imperative Approach to Temporal Logic Programming. *Formal Aspects of Computing*, 7(E) pp. 111-155.

[BFG89b] Barringer, H., Fisher, M. and Gough, G.: Fair SMG and Linear Time Model Checking. In *Proceedings of Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 1989. (Published in *Lecture Notes in Computer Science*, volume 407, Springer-Verlag).

[BFG91] Barringer, H., Fisher, M., Gabbay, D. and Hunter, A.: Meta-Reasoning in Executable Temporal Logic. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Cambridge, Massachusetts, April 1991. Morgan Kaufmann.

[BG88] Barringer, H. and Gabbay, D.: Executing temporal logic: Review and prospects (Extended Abstract). In *Proceedings of Concurrency '88*, 1988.

[Brz95] Brzoska, C.: Temporal Logic Programming with Metric and Past Operators. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Hiedelberg, Germany, 1995.

[EmC82] Emerson, E. A. and Clarke, E. M.: Using Branching Time Temporal Logic to Synthesise Synchronisation Skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[FiB91] Fisher, M. and Barringer, H.: Concurrent METATEM Processes — A Language for Distributed AI. In *Proceedings of the European Simulation Multiconference*, Copenhagen, Denmark, June 1991.

[FFO93] Finger, M., Fisher, M. and Owens, R.: METATEM at Work: Modelling Reactive Systems Using Executable Temporal Logic. In *Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE)*, Edinburgh, U.K., June 1993. Gordon and Breach Publishers.

[Fis92] Fisher, M.: A Normal Form for First-Order Temporal Formulae. In *Proceedings of Eleventh International Conference on Automated Deduction (CADE)*, Saratoga Springs, New York, June 1992. (Published in *Lecture Notes in Computer Science*, volume 607, Springer-Verlag).

[Fis93] Fisher, M.: Concurrent METATEM — A Language for Modeling Reactive Systems. In *Parallel Architectures and Languages, Europe (PARLE)*, Munich, Germany, June 1993. Springer-Verlag.

[Fis94] Fisher, M.: A Survey of Concurrent METATEM — The Language and its Applications. In *First International Conference on Temporal Logic (ICTL)*, Bonn, Germany, July 1994. (Published in *Lecture Notes in Computer Science*, volume 827, Springer-Verlag).

[FKT86] Fujita, M., Kono, S., Tanaka, T. and Moto-oka, T.: Tokio: Logic Programming Language based on Temporal Logic and its compilation into Prolog. In *3rd International Conference on Logic Programming*, London, July 1986. (Published in *Lecture Notes in Computer Science*, volume 225, Springer-Verlag).

[FMO91] Finger, M., McBrien, P. and Owens, R.: Databases and Executable Temporal Logic. In *Proceedings of the ESPRIT Conference*, November 1991.

[FiN92] Fisher, M. and Noël, P.: Transformation and Synthesis in METATEM – Part I: Propositional METATEM. Technical Report UMCS-92-2-1, Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, U.K., February 1992.

[FiO92] Fisher, M. and Owens, R.: From the Past to the Future: Executing Temporal Logic Programs. In *Proceedings of Logic Programming and Automated Reasoning (LPAR)*, St. Petersberg, Russia, July 1992. (Published in *Lecture Notes in Computer Science*, volume 624, Springer-Verlag).

[FiO95a] Fisher, M. and Owens, R.: An Introduction to Executable Modal And Temporal Logics. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Hiedelberg, Germany, 1995.

[FiO95b] Fisher, M. and Owens, M.: editors. *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, February 1995.

[Frü95] Frühwirth, T.: Temporal Logic and Annotated Constraint Logic Programming. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Hiedelberg, Germany, 1995.

* This paper can be retrieved by downloading the (compressed PostScript) file FACj_7E_pp111–155.ps.Z which can be found in the directory pub/fac of the ftp.cs.ac.uk.

[FiW93] Fisher, M. and Wooldridge, M.: Executable Temporal Logic for Distributed A.I. In *Twelfth International Workshop on Distributed A.I.*, Hidden Valley Resort, Pennsylvania, May 1993.

[Gab87a] Gabbay, D.: Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Colloquium on Temporal Logic in Specification*, Altrincham, U.K., 1987. (Published in *Lecture Notes in Computer Science*, volume 398, Springer-Verlag).

[Gab91] Gabbay, D.: Modal and Temporal Logic II (A Temporal Prolog Machine). In T. Dodd, R. Owens, and S. Torrance, editors, *Logic Programming—Expanding the Horizon*. Intellect Books Ltd, 1991.

[Gou84] Gough, G. D.: Decision Procedures for Temporal Logic. Master's thesis, Department of Computer Science, University of Manchester, October 1984.

[Hal87] Hale, R.: Temporal Logic Programming. In A. Galton, editor, *Temporal Logics and their Applications*, chapter 3, pages 91–119. Academic Press, London, December 1987.

[HaM87] Hale, R. and Moszkowski, B.: Parallel Programming in Temporal Logic. In *Parallel Architectures and Languages Europe (PARLE)*, Eindhoven, The Netherlands, June 1987. (Published as Lecture Notes in Computer Science, volume 259, Springer Verlag, Berlin.).

[Hry88] Hrycej, T.: Temporal Prolog. In Yves Kodratoff, editor, *Proceedings of the European Conference on Artificial Intelligence (ECAI)*. Pitman Publishing, August 1988.

[Hry93] Hrycej, T.: A temporal extension of Prolog. *The Journal of Logic Programming*, 15(1 & 2):113–145, January 1993.

[JaL87] Jaffir, J. and Lassez, J-L.: Constraint Logic Programming. In *Proceedings of the Fourteenth ACM Symposium on the Principles of Programming Languages*, pages 111–119, Munich, West Germany, January 1987.

[LPZ85] Lichtenstein, O., Pnueli, A. and Zuck, L.: The Glory of the Past. *Lecture Notes in Computer Science*, 193:196–218, June 1985.

[Mer95] Merz, S.: Efficiently Executable Temporal Logic Programs. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Hiedelberg, Germany, 1995.

[Mos83] Moszkowski, B.: Reasoning about digital circuits. Technical report, Stanford University, 1983.

[Mos86] Moszkowski, B.: *Executing Temporal Logic Programs*. Cambridge University Press, Cambridge, UK, 1986.

[MaW84] Manna, Z. and Wolper, P.: Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1), January 1984.

[Noë91] Noël, P.: A Method for the Determinisation of Propositional Temporal Formulae. In T. Clement and K. Lau, editors, *Proceedings of Workshop on Logic Program Synthesis and Transformation (LOPSTR)*, Manchester, UK, July 1991. Springer-Verlag.

[OrM94] Orgun, M. and Ma, W.: An Overview of Temporal and Modal Logic Programming. In *First International Conference on Temporal Logic (ICTL)*, Bonn, Germany, July 1994. (Published in *Lecture Notes in Computer Science*, volume 827, Springer-Verlag).

[OrW92a] Orgun, M. and Wadge, W.: Theory and Practice of Temporal Logic Programming. In L Fariñas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*. Oxford University Press, 1992.

[OrW92b] Orgun, M. and Wadge, W.: Towards a unified theory of intensional logic programming. *The Journal of Logic Programming*, 13(1, 2, 3 and 4):413–440, 1992.

[PnR89a] Pnueli, A. and Rosner, R.: On the Synthesis of a Reactive Module. In *Proceedings of the Sixteenth ACM Symposium on the Principles of Programming Languages (POPL)*, pages 179–190, 1989.

[PnR89b] Pnueli, A. and Rosner, R.: On the Synthesis of an Asynchronous Reactive Module. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages and Programs (ICALP)*, 1989.

[Tan89] Tang, T.: Temporal Logic CTL + Prolog. *Journal of Automated Reasoning*, 5:49–65, 1989.

[ToM90] Torsun, I. S. and Manning, K. J.: Execution and Application of Temporal Modal Logic. Internal Report, Department of Computing, University of Bradford, U.K., 1990. (Part of the Alvey final report on the Logic Database Demonstrator Project).

Received July 1994
Accepted in revised form June 1995 by D. J. Cooke