Implementing Temporal Logics: Tools for Execution and Proof

[EXTENDED ABSTRACT]

Michael Fisher

Logic and Computation Group, Department of Computer Science
University of Liverpool, Liverpool L69 7ZF
[M.Fisher@csc.liv.ac.uk http://www.csc.liv.ac.uk/∼michael]

1 Introduction

Here we present an overview of a selection of tools for execution and proof based on temporal logic, and will outline both the general techniques used and problems encountered in implementing them. The tools considered will mainly be theorem-provers and (logic-based) programming languages. Specifically:

– clausal temporal resolution [13, 18] and its implementation as Clatter [10], TRP [28, 27] and TRP++ [26];
– executable temporal logics [4, 15] and its implementation as METATEM [3], Concurrent METATEM [14, 31] and MAGENTA [17, 19, 21, 20].

In addition, we will briefly mention induction-based temporal proof [5], temporal logic programming [1], and model checking [7].

Rather than providing detailed algorithms, this presentation will concentrate on general principles, outlining current problems and future possibilities.

Acknowledgements Much of this work has been supported by funding from the Engineering and Physical Sciences Research Council (EPSRC)[1], and has involved collaboration with a variety of outstanding colleagues from within the Logic and Computation group[2]. I thank them all for their endeavours.

2 What is Temporal Logic?

2.1 Some History

Temporal logic was originally developed in order to represent tense in natural language [36]. Within Computer Science, it has achieved a significant role in the formal specification and verification of concurrent and distributed systems [35]. Much of this popularity has been achieved as a number of useful concepts, such as *safety*, *liveness* and *fairness* can be formally, and concisely, specified using temporal logics [12, 33].

[1] `http://www.epsrc.ac.uk`
[2] `http://www.csc.liv.ac.uk/research/logics`

2.2 Some Intuition

In their simplest form, temporal logics can be seen as extensions of classical logic, incorporating additional operators relating to time. These operators are typically: '$\bigcirc$', meaning "in the next moment in time", '$\square$', meaning "at every future moment", and '$\diamondsuit$', meaning "at some future moment". These additional operators allow us to express statements such as

$$\square(\mathit{send} \Rightarrow \diamondsuit \mathit{received})$$

to characterise the statement

> "it is always the case that if we send a message then, at some future moment it *will* be received".

The flexibility of temporal logic allows us to use formulae such as

$$\square(\mathit{send} \Rightarrow \bigcirc(\mathit{received} \vee \mathit{send}))$$

which is meant to characterise

> "it is always the case that, if we send a message then, at the next moment in time, either the message will be received or we will send it again"

and

$$\square(\mathit{received} \Rightarrow \neg \mathit{send})$$

meaning

> "it is always the case that if a message is received it cannot be sent again".

Thus, given formulae of the above form then, if we try to send a message, i.e. '*send*', we *should* be able to show that it is *not* the case that the system continually re-sends the message (but it is never received) i.e. the statement

$$\square \mathit{send} \wedge \square \neg \mathit{received}$$

should be inconsistent.

2.3 Some Applications

The representation of dynamic activity via temporal formalisms is used in a wide variety of areas within Computer Science and Artificial Intelligence, for example Temporal Databases, Program Specification, System Verification, Agent-Based Systems, Robotics, Simulation, Planning, Knowledge Representation, and many more. While we are not able to describe all these aspects here, the interested reader should see, for example, [33, 34, 7, 41, 29].

There are many different temporal logics (see, for example [12]). The models of time which underly these logics can be discrete, dense or interval-based, linear, branching or partial order, finite or infinite, etc. In addition, the logics can have a wide range of operators, such as those related to discrete future-time (e.g: $\bigcirc$, $\Diamond$, $\Box$), interval future-time (e.g: $\mathcal{U}$, $\mathcal{W}$), past-time (e.g: $\bullet$ $\blacklozenge$, $\blacksquare$, $\mathcal{S}$, $\mathcal{Z}$), branching future-time (e.g: **A**, **E**), fixed-point generation (e.g: μ, ν) and propositional, quantified propositional or full first-order variations. Even then, such temporal logics are often combined with standard modal logics. For example, typical combinations involve TL + S5 modal logic (often representing 'Knowledge'), or TL + KD45 (Belief) + KD (Desire) + KD (Intention).

Here, we will mainly concentrate on one very popular variety, namely discrete linear temporal logic, which has an underlying model of time isomorphic to the Natural Numbers (i.e. an infinite sequence with distinguished initial point) and is also linear, with each moment in time having at most one successor. (Note that the infinite and linear constraints ensure that each moment in time has *exactly* one successor, hence the use of a single '$\bigcirc$' operator.)

3 Where's the Difficulty?

Temporal Logics tend to be complex. To give some intuition why this is the case, let us look at a few different ways of viewing (propositional) temporal logic.

Propositional temporal logic can be thought of as

1. *a specific decidable (PSPACE-complete) fragment of classical first-order logic*
   For example, the semantics of (discrete, linear) propositional temporal logic can be given by translation as:
   $i \models \bigcirc p \quad \rightarrow \quad p(i+1)$
   $i \models \Diamond p \quad \rightarrow \quad \exists j.\ (j \geq i) \wedge p(j)$
   $i \models \Box p \quad \rightarrow \quad \forall j.\ (j \geq i) \Rightarrow p(j)$
   [this can be a problem as proof/execution techniques often find it hard to isolate exactly this fragment]

2. *a multi-modal logic, comprising two modalities,* $[1]$ *and* $[*]$, *which interact closely*
   The induction axiom in discrete temporal logic

$$\vdash\ \Box(\varphi \Rightarrow \bigcirc\varphi)\ \Rightarrow\ (\varphi \Rightarrow \Box\varphi)$$

   can be viewed as the *interaction* axiom between modalities

$$\vdash\ [*](\varphi \Rightarrow [1]\varphi)\ \Rightarrow\ (\varphi \Rightarrow [*]\varphi)$$

   Thus, $[1]$ is usually represented as '$\bigcirc$', while $[*]$ is usually represented as '$\Box$'
   [while mechanising modal logics is relatively easy, multi-modal problems become complex when interactions occur between the modalities; in our case the interaction is of an *inductive* nature]

3. *a characterisation of simple induction*
   The induction axiom in discrete temporal logic

$$\vdash\ \Box(\varphi \Rightarrow \bigcirc\varphi)\ \Rightarrow\ (\varphi \Rightarrow \Box\varphi)$$

can alternatively be viewed as

$$[\forall i.\ \varphi(i) \Rightarrow \varphi(i+1)] \ \Rightarrow\ [\varphi(0) \Rightarrow \forall j.\ \varphi(j)]$$

[again, this use of induction can cause problems]

4. *a logic over sequences, trees or partial-orders (depending on the model of time)*
   For example, a sequence-based semantics can be given for discrete linear temporal logic:

   $s_i, s_{i+1}, \ldots, s_\omega \models \bigcirc p$ if, and only if, $s_{i+1}, \ldots, s_\omega \models p$
   $s_i, s_{i+1}, \ldots, s_\omega \models \Diamond p$ if, and only if, there exists a $j \geq i$ such that $s_j, \ldots, s_\omega \models p$
   $s_i, s_{i+1}, \ldots, s_\omega \models \Box p$ if, and only if, for all $j \geq i$ then $s_j, \ldots, s_\omega \models p$

   [this shows that temporal logic can be used to characterise a great variety of, potentially complex, computational structures]

5. *a syntactic characterisation of finite-state automata over infinite words (ω-automata)*
   For example
   – formulae such as $p \Rightarrow \bigcirc q$ give constraints on possible state transitions,
   – formulae such as $p \Rightarrow \Diamond r$ give constraints on accepting states within an automaton, and
   – formulae such as $p \Rightarrow \Box s$ give global constraints on states in an automaton.

   [this shows some of the power of temporal logic as a variety of different ω-automata can be characterised in this way]

3.1 A Little Complexity

The decision problem for a simple propositional (discrete, linear) temporal logic is already PSPACE-complete [37]; other variants of temporal logic may be worse! When we move to *first-order* temporal logics, things begin to get unpleasant. It is easy to show that first-order temporal logic is, in general, incomplete (i.e. not recursively-enumerable [38, 2]). In fact, until recently, it has been difficult to find *any* non-trivial fragment of first-order temporal logic that has reasonable properties. A breakthrough by Hodkinson *et. al.* [23] showed that *monodic* fragments of first-order temporal logic could be complete, even decidable.

4 What Tools?

The main tools that we are interested in are used to carry out *temporal verification*, via resolution on temporal formulae, and *temporal execution*, via direct execution of temporal formulae. In our case, both of these use temporal formulae within a specific normal form, called *Separated Normal Form (SNF)* [16].

4.1 SNF

A temporal formula in Separated Normal Form (SNF) is of the form

$$\Box \bigwedge_{i=1}^{n} (P_i \Rightarrow F_i)$$

where each of the '$P_i \Rightarrow F_i$' (called *clauses* or *rules*) is one of the following

$$\begin{array}{ll}
\textbf{start} \Rightarrow \bigvee_{k=1}^{r} l_k & \text{(an initial clause)} \\
\bigwedge_{j=1}^{q} m_j \Rightarrow \bigcirc \bigvee_{k=1}^{r} l_k & \text{(a step clause)} \\
\bigwedge_{j=1}^{q} m_j \Rightarrow \Diamond l & \text{(a sometime clause)}
\end{array}$$

where each l, l_k or m_j is a literal and '**start**' is a formula that is only satisfied at the "beginning of time".

Thus, the intuition is that:

- initial clauses provide *initial* constraints;
- step clauses provide constraints on the *next* step; and
- sometime clauses provide constraints on the *future*.

We can provide simple examples showing some of the properties that might be represented directly as SNF clauses.

- Specifying initial conditions: $\textbf{start} \Rightarrow \textit{sad}$
- Defining transitions between states: $(\textit{sad} \wedge \neg \textit{rich}) \Rightarrow \bigcirc \textit{sad}$
- Introducing new eventualities (goals): $(\neg \textit{resigned} \wedge \textit{sad}) \Rightarrow \Diamond \textit{famous}$
  $\textit{sad} \Rightarrow \Diamond \textit{happy}$
- Introducing permanent properties: $\textit{lottery-win} \Rightarrow \bigcirc \Box \textit{rich}$ which, in SNF, becomes

$$\begin{array}{r}
\textit{lottery-win} \Rightarrow \bigcirc \textit{rich} \\
\textit{lottery-win} \Rightarrow \bigcirc x \\
x \Rightarrow \bigcirc \textit{rich} \\
x \Rightarrow \bigcirc x
\end{array}$$

Translation from an arbitrary temporal formula into SNF is an operation of polynomial complexity [16, 18].

We also need the concept of a *merged* SNF clause: any SNF clause is a merged SNF clause and, given two merged SNF clauses $A \Rightarrow \bigcirc B$ and $C \Rightarrow \bigcirc D$, we can generate a new merged SNF clause $(A \wedge C) \Rightarrow \bigcirc (B \wedge D)$.

4.2 Clausal Resolution

Given a set of clauses in SNF, we can apply resolution rules, such as

$$\text{Initial Resolution:}\quad \frac{\begin{array}{l}\mathbf{start} \Rightarrow (A \vee l)\\ \mathbf{start} \Rightarrow (B \vee \neg l)\end{array}}{\mathbf{start} \Rightarrow (A \vee B)}$$

$$\text{Step Resolution:}\quad \frac{\begin{array}{r}P \Rightarrow \bigcirc(A \vee l)\\ Q \Rightarrow \bigcirc(B \vee \neg l)\end{array}}{(P \wedge Q) \Rightarrow \bigcirc(A \vee B)}$$

$$\text{Temporal Resolution (simplified)}^{3}\text{:}\quad \frac{\begin{array}{l}A \Rightarrow \bigcirc\Box\neg l\\ Q \Rightarrow \Diamond l\end{array}}{Q \Rightarrow (\neg A)\,\mathcal{W}\,l}$$

As we will see later, it is this *temporal resolution* rule that causes much of the difficulty.

4.3 Executable Temporal Logics

We use the *Imperative Future* approach [4]:

- transforming the temporal specification into SNF;
- from the initial constraints, *forward chaining* through the set of temporal rules representing the agent; and
- constraining the execution by attempting to satisfy goals, such as $\Diamond g$ (i.e. g eventually becomes true).

Since some goals might not be able to be satisfied immediately, we must keep track of the outstanding goals and reconsider them later. The basic strategy used is to attempt to satisfy the oldest outstanding eventualities first and keep a record of the others, retrying them as execution proceeds.

Example Imagine a 'car' agent which can *go*, *turn* and *stop*, but can also run out of fuel (*empty*) and *overheat*.

The agent's internal definition might be given by a temporal logic specification in SNF, for example,

$$\begin{array}{rcl}\mathbf{start} & \Rightarrow & \neg moving\\ go & \Rightarrow & \Diamond moving\\ (moving \wedge go) & \Rightarrow & \bigcirc(overheat \vee empty)\end{array}$$

The car agent's behaviour is implemented by *forward-chaining* through these formulae.

- Thus, *moving* is false at the beginning of time.
- Whenever *go* is true, a commitment to eventually make *moving* true is given.

[3] $(\neg A)\,\mathcal{W}\,l$ is satisfi ed either if $\neg A$ is always satisfi ed, or if $\neg A$ is satisfi ed up to a point when l is satisfi ed.

– Whenever both *go* and *moving* are true, then either *overheat* or *empty* will be made true in the next moment in time.

This provides the basis for temporal execution, and has been extended with execution for combinations with *modal* logics, deliberation mechanisms [17], resource-bounded reasoning [19] and a concurrent operational model [14].

5 Implementations

5.1 Clausal Temporal Resolution

The essential complexity in carrying out clausal temporal resolution is implementing the temporal resolution rule itself. First, let us note that the Temporal Resolution rule outlined earlier is not in the correct form. The *exact* form of this rule is

$$\text{Temporal Resolution (full):}\quad \frac{\begin{array}{l} A_1 \Rightarrow \bigcirc B_1 \\ \ldots \Rightarrow \ldots \\ A_n \Rightarrow \bigcirc B_n \\ Q \Rightarrow \Diamond l \end{array}}{Q \Rightarrow (\bigwedge_{i=1}^{n} \neg A_i)\, \mathcal{W}\, l}$$

where each $A_i \Rightarrow \bigcirc B_i$ is a merged SNF clause and each B_i satisfies $B_i \Rightarrow (\neg l \wedge \bigvee_{j=1}^{n} A_j)$.

Thus, in order to implement this rule, a *set* of step clauses satisfying certain properties must be found; such a set is called a *loop*. This process has undergone increasing refinement, as has the implementation of clausal temporal resolution provers in general:

1. The original approach proposed was to conjoin all sets of step clauses to give, so called, *merged* SNF clauses and then treat these merged clauses as edges/transitions in a graph. Finding a loop is then just a question of extracting a *strongly connected component* from the graph, which is a linear operation [39].

   The problem here is explicitly constructing the large set of merged SNF clauses.

2. Dixon [8–10] developed an improved (breadth-first) search algorithm, which formed the basis of the '`Clatter`' prover. This search approach effectively aimed to generate only the merged SNF clauses that were required to find a loop, rather than generating all such clauses.

   The problem with the `Clatter` family of provers was the relatively slow link to the classical resolution system (which was used to carry out the step resolution operations).

3. Hustadt then developed `TRP` [28]. The idea here was to use arithmetical translations to translate as much as possible of the process to classical resolution operations and then use an efficient classical resolution system. In addition, `TRP` used a translation of the breadth-first loop search algorithm into a series of classical resolution problems suggested in [11]. (`TRP` is also able to deal with the combination

of propositional temporal logic with various modal logics including KD45 and S5.) The resulting system was evaluated against other decision procedures for this form of temporal logic and was shown to be very competitive [28, 27].

   The main problems with `TRP` were that it was implemented in `Prolog` and that the data/term representation/indexing techniques could be improved.

4. The latest variety of clausal temporal resolution system is `TRP++`, implemented by Konev [26]. Here, `TRP` is re-implemented in `C++` and is refined with a number of contemporary data representation and indexing techniques.

   `TRP++` currently performs very well in comparison with other provers for propositional temporal logic.

5.2 Executable Temporal Logics

The Imperative Future style of execution provides a relatively simple approach to executing temporal logic formulae. As described above, beginning at the initial conditions we simply forward chain through the step clauses/rules generating a model, all the time constraining the execution with formulae such as '$\Diamond g$'.

The development in this area has not primarily been concerned with speed. As we will see below, the developments has essentially involved refining and extending the internal capabilities of the programs and allowing for more complex interactions between programs.

Thus, the implementations of this approach, beginning with METATEM, proceeded as follows.

1. The first approach, reported in [22], essentially used a `Prolog` meta-interpreter to implement the system. The forward chaining aspect is relatively standard, and the management of outstanding eventualities (i.e. formulae such as '$\Diamond g$') was handled with a queue structure.

   In order to maintain completeness (in the propositional case) a form of *past-time loop checking* had to be employed. This involved retaining a large proportion, and sometimes *all*, of the history of the computation and checking for loops over this as every new computation state was constructed. (Note that this loop-checking aspect is usually omitted from the later languages.)

   The main problems with this approach were the lack of features, particularly those required for programming rational agents, such as internal reasoning, deliberation and concurrency.

2. In [14], Concurrent METATEM was developed. This allowed for multiple asynchronous, communicating METATEM components and provided a clean interaction between the internal (logical) computation and the concurrent operational model.

   Concurrent METATEM was implemented in `C++` but was relatively slow and static (i.e. process scheduling was implemented directly).

3. Kellett, in [30, 31], developed more refined implementation techniques for Concurrent METATEM. Here, individual METATEM programs were compiled into (linked) pairs of I/O automata [32], one to handle the internal computation, the other to handle the interaction with the environment. Such automata can then, potentially, be cloned (for process spawning) and moved (for load balancing and mobility).

   While Concurrent METATEM provides an interesting model of simple multi-agent computation, work was still required on the internal computation mechanism for each individual agent.

4. More recently, the internal computation has been extended by providing a *belief* dimension, allowing meta-control of the deliberation[4], allowing resource-bounded reasoning and incorporating agent abilities [17, 19, 20].

   This has led to the MAGENTA language in which rational agents can be implemented, and in which complex multi-agent organisations can be developed. Currently, the internal MAGENTA implementation is provided by `Prolog`, but work is under way to provide a `Java` implementation of both individual and group aspects [21].

5.3 Other Techniques

In this section, we will briefly mention a few other systems related to temporal logic that we are working on.

Induction-based Temporal Proof As mentioned above, first-order temporal logics are complex. In particular, full first-order temporal logic is not recursively-enumerable. However, as we still wish to prove theorems within such a logic, we have been developing techniques to support this. Such a system is described in [5], where an induction-based theorem-prover is enhanced with heuristics derived from the clausal temporal resolution techniques (see above) and implemented in λClam/λProlog.

Temporal Logic Programming Standard logic programming techniques were transferred to temporal logic in [1]. However, because of the incompleteness of first-order temporal logics, the language was severely restricted. In fact, if we think of SNF above then the fragment considered is essentially that consisting of initial and step clauses. Thus, implementation for such a language is a small extension of classical logic programming techniques and constraint logic programming techniques.

Model Checking Undoubtedly the most popular application of temporal logic is in *model checking*. Here, a finite-state model capturing the executions of a system is checked against a temporal formula. These finite state models often capture hardware descriptions, network protocols or complex software [24, 7]. While much model-checking technology was based on automata-theoretic techniques, advances in *symbolic* [6] and *on-the-fly* [25] techniques have made model checking the success it is. Current work on abstraction techniques and Java model checking, such as [40], promise even greater advances.

[4] Deliberation here means the process of deciding in which order to attempt outstanding eventualities at each computation step.

6 Summary

We have overviewed a selection of tools for execution and proof within temporal logic. Although these tools are generally prototypes, they are increasingly used in realistic scenarios, and more sophisticated versions appear likely to have a significant impact in both Computer Science and Artificial Intelligence.

References

1. M. Abadi and Z. Manna. Temporal Logic Programming. *Journal of Symbolic Computation*, 8: 277–295, 1989.
2. M. Abadi. The Power of Temporal Proofs. *Theoretical Computer Science*, 64:35–84, 1989.
3. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: An Introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.
4. H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, editors. *The Imperative Future: Principles of Executable Temporal Logics*. Research Studies Press, Chichester, United Kingdom, 1996.
5. J. Brotherston, A. Degtyarev, M. Fisher, and A. Lisitsa. Searching for Invariants using Temporal Resolution. In *Proceedings of LPAR-2002*. Springer Verlag, 2002. LNCS, to appear.
6. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. In *Proceedings of the 5th Symposium on Logic in Computer Science, Philadelphia*, June 1990.
7. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.
8. C. Dixon. *Strategies for Temporal Resolution*. PhD thesis, Department of Computer Science, University of Manchester, Manchester M13 9PL, U.K., December 1995.
9. C. Dixon. Search Strategies for Resolution in Temporal Logics. In *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE)*, volume 1104 of *LNCS*, New Brunswick, New Jersey, July/August 1996. Springer-Verlag.
10. C. Dixon. Temporal Resolution using a Breadth-First Search Algorithm. *Annals of Mathematics and Artifi cial Intelligence*, 22:87–115, 1998.
11. C. Dixon. Using Otter for Temporal Resolution. In *Advances in Temporal Logic*. Kluwer Academic Publishers, 1999.
12. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.
13. M. Fisher. A Resolution Method for Temporal Logic. In *Proc. Twelfth International Joint Conference on Artifi cial Intelligence (IJCAI)*, Sydney, Australia, 1991. Morgan Kaufman.
14. M. Fisher. Concurrent METATEM — A Language for Modeling Reactive Systems. In *Parallel Architectures and Languages, Europe (PARLE)*, Munich, Germany, June 1993. (Published in *LNCS*, volume 694, Springer-Verlag).
15. M. Fisher. Representing and Executing Agent-Based Systems. In *Intelligent Agents*. Springer-Verlag, 1995.
16. M. Fisher. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4):429–456, August 1997.
17. M. Fisher. Implementing BDI-like Systems by Direct Execution. In *Proceedings of International Joint Conference on Artifi cial Intelligence (IJCAI)*. Morgan-Kaufmann, 1997.
18. M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, January 2001.
19. M. Fisher and C. Ghidini. Programming Resource-Bounded Deliberative Agents. In *Proceedings of International Joint Conference on Artifi cial Intelligence (IJCAI)*. Morgan Kaufmann, 1999.

20. M. Fisher, C. Ghidini, and T. Kakoudakis. The ABC of Rational Agent Programming. In *Proc. First International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2002.
21. M. Fisher and T. Kakoudakis. Flexible Agent Grouping in Executable Temporal Logic. In *Proceedings of Twelfth International Symposium on Languages for Intensional Programming (ISLIP)*. World Scientifi c Press, 1999.
22. M. Fisher and R. Owens. 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 *LNCS*, volume 624, Springer-Verlag).
23. I. Hodkinson, F. Wolter, and M. Zakharyashev. Decidable fragments of fi rst-order temporal logics. *Annals of Pure and Applied Logic*, 2000.
24. G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
25. G.J. Holzmann. The model checker spin. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.
26. U. Hustadt and B. Konev. TRP++: A Temporal Resolution prover. (Submitted), 2002.
27. U. Hustadt and R. A. Schmidt. Scientifi c benchmarking with temporal logic decision procedures. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eighth International Conference (KR'2002)*, pages 533–544. Morgan Kaufmann, 2002.
28. U. Hustadt and R.A. Schmidt. Formulae which highlight differences between temporal logic and dynamic logic provers. In *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, Technical Report DII 14/01, pages 68–76. Dipartimento di Ingegneria dell'Informazione, Unversitá degli Studi di Siena, 2001.
29. M. Huth and M. Ryan. *Logic in Computer Science*. Cambridge University Press, 2000.
30. A. Kellett. *Implementation Techniques for Concurrent* METATEM. PhD thesis, Department of Computing and Mathematics, Manchester Metropolitan University, 2000.
31. A. Kellett and M. Fisher. Automata Representations for Concurrent METATEM. In *Proceedings of the Fourth International Workshop on Temporal Representation and Reasoning (TIME)*. IEEE Press, May 1997.
32. N. A. Lynch and M. R. Tuttle. An Introduction to Input/Output Automata. Technical Report MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology, November 1988.
33. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specifi - cation*. Springer-Verlag, New York, 1992.
34. Z. Manna and A. Pnueli. *Temporal Verifi cation of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
35. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the Eighteenth Symposium on the Foundations of Computer Science*, Providence, USA, November 1977.
36. A. Prior. *Past, Present and Future*. Oxford University Press, 1967.
37. A. P. Sistla and E. M. Clarke. Complexity of propositional linear temporal logics. *ACM Journal*, 32(3):733–749, July 1985.
38. A. Szalas and L. Holenderski. Incompleteness of First-Order Temporal Logic with Until. *Theoretical Computer Science*, 57:317–325, 1988.
39. R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.
40. W. Visser, K. Havelund, G. Brat, and S. Park. Model checking programs. In *International Conference on Automated Software Engineering (ASE)*, September 2000.
41. M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.