Execution and Proof in a Horn-Clause Temporal Logic

Clare Dixon[1], Michael Fisher[1] and Mark Reynolds[2]

[1] Department of Computing, Manchester Metropolitan University, Manchester M1 5GD, U.K.
{C.Dixon,M.Fisher}@doc.mmu.ac.uk

[2] Department of Computer Science, King's College, Strand, London, WC2R 2LS, U.K.
markr@dcs.kcl.ac.uk

Abstract. Both proof, via clausal resolution, and execution, via the imperative future approach, depend on the use of a normal form for temporal formulae. While the systems developed have centred around the use of an unrestricted normal form, we here consider a Horn clause-like version of the normal form and its effect on both execution and resolution. This refined normal form is as expressive as the original, and represents a natural way to describe systems, yet allows both execution and resolution to be implemented more efficiently in practice.

1 Introduction

The direct execution of temporal logic formulae has become increasingly popular over recent years [Mos86, dCP92, FO95], with a range of different logics being executed using a variety of programming paradigms. In many of these cases the expressive power of the programming notation is restricted in order to allow efficient implementation. In the METATEM system [BFG$^+$95], however, *any* formula within the underlying temporal logic can be executed. This leads to a situation where complex temporal formulae can be defined and where execution of such formulae may be particularly costly.

Whilst execution is the animation of a temporal logic formula or *specification*, *verification* that a temporal logic specification satisfies a particular property requires proof. Thus proof methods for temporal logics have also been developed based on, for example, automata [VW86], tableau [Wol85] and resolution [Aba87, Fis91]. Here we consider the resolution based method given in [Fis91]. This clausal resolution based method for propositional linear-time temporal logics (PTL) applies classical style resolution to formulae occurring at the same moment in time and temporal resolution between formulae that state a condition must occur sometime in the future with formulae that ensure that the condition can never occur.

The basis of both METATEM and the temporal resolution method is a normal form, Separated Normal Form (SNF). Whilst SNF has been defined for both propositional [Fis91] and first-order [Fis97] linear-time temporal logics we limit our discussion to the propositional version as the first-order version is beyond the scope of this paper. The translation of any PTL formula into SNF reduces the number of different temporal operators to a core set and requires formulae to be in a particularly simple 'rule' form. If the original formula is satisfiable then so is the resulting set of formulae.

In classical logic it is usual to translate into a normal form, resulting in a set of *clauses*, before applying any resolution rules [Rob65]. If the normal form is restricted such that only clauses with at most one positive literal are allowed (Horn clauses) then efficient strategies such as unit [WCR64] or input [Cha70] resolution may be used to guide the proof whilst retaining completeness. Similarly for execution, the programming language Prolog requires Horn clauses. Thus, in this paper we investigate (temporal) Horn clause-like restrictions on SNF, analysing their effect on both execution and resolution. This paper is structured as follows. The syntax and semantics of PTL are outlined in §2 and SNF and its restricted version RSNF are presented. The complexity of RSNF is discussed in §3. In §4 a review of METATEM and temporal resolution is given. The effect of restricting SNF upon such execution and proof is considered in §5 and §6 respectively. Concluding remarks are made in §7.

2 A Linear Temporal Logic

2.1 Syntax and Semantics

The logic used in this paper is Propositional Temporal Logic (PTL), in which we use a linear, discrete model of time with finite past and infinite future [GPSS80]. PTL may be viewed as a classical propositional logic augmented with both future-time and past-time temporal operators. Future-time temporal operators include '$\Diamond$' (*sometime in the future*), '$\Box$' (*always in the future*), '$\bigcirc$' (*in the next moment in time*), '$\mathcal{U}$' (*until*), '$\mathcal{W}$' (*unless* or *weak until*), each with a corresponding past-time operator. Since our temporal models assume a finite past, for convenience, we define an operator, **start**, that is only true at the beginning of time.

Models for PTL consist of a sequence of *states*, representing moments in time, i.e.,

$$\sigma = s_0, s_1, s_2, s_3, \ldots$$

Here, each state, s_i, contains those propositions satisfied in the i^{th} moment in time. As formulae in PTL are interpreted at a particular moment, the satisfaction of a formula f is denoted by $(\sigma, i) \models f$, where σ is the model and i is the state index at which the temporal statement is to be interpreted. For any well-formed formula f, model σ and state index i, then either $(\sigma, i) \models f$ or $(\sigma, i) \not\models f$. For example, a proposition symbol, 'p', is satisfied in model σ and at state index i if, and only if, p is one of the propositions in state s_i, i.e.,

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

The semantics of the future-time temporal connectives are defined as follows

$(\sigma, i) \models$ **start** iff $i = 0$;
$(\sigma, i) \models \bigcirc A$ iff $(\sigma, i+1) \models A$;
$(\sigma, i) \models \Diamond A$ iff there exists a $j \geq i$ s.t. $(\sigma, j) \models A$;
$(\sigma, i) \models \Box A$ iff for all $j \geq i$ then $(\sigma, j) \models A$;
$(\sigma, i) \models A \mathcal{U} B$ iff there exists a $k \geq i$ s.t. $(\sigma, k) \models B$ and for all $i \leq j < k$ then $(\sigma, j) \models A$;
$(\sigma, i) \models A \mathcal{W} B$ iff $(\sigma, i) \models A \mathcal{U} B$ or $(\sigma, i) \models \Box A$.

2.2 A Normal Form for PTL

Formulae in PTL can be transformed in to Separated Normal Form (SNF), which is used as the basis for both execution, via the METATEM language [BFG+95], and proof, via a clausal resolution method [Fis91]. SNF was introduced first in [Fis91] and has been extended to first-order temporal logic in [Fis97]. Here we use a future-time version of SNF which is equivalent to the original (which contained both future and past-time formulae). While the translation from an arbitrary temporal formula to SNF will not be described here, we note that such a transformation not only preserves satisfiability, but also any model generated from the formula in SNF is a model for the original formula. A formula in (future-time) SNF is of the form:

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

where each of the '$\phi_i \Rightarrow \psi_i$' (called *rules*) is one of the following

$$\mathbf{start} \quad \Rightarrow \bigvee_{k=1}^{r} l_k \qquad \text{(an \textit{initial} rule)}$$

$$\bigwedge_{j=1}^{q} m_j \Rightarrow \bigcirc \bigvee_{k=1}^{r} l_k \qquad \text{(an \textit{always} rule)}$$

$$\bigwedge_{j=1}^{q} m_j \Rightarrow \Diamond l \qquad \text{(a \textit{sometime} rule)}$$

and each m_j, l_k, and l are literals. In certain situations we require sets of the always rules to be combined or *merged* where their left hand sides are conjoined and simplified and the right hand sides are conjoined, simplified and rewritten in DNF. Such merged rules are known as SNF$_m$ rules.

2.3 Restricted SNF (RSNF)

In the rest of the paper we further restrict the normal form and analyse the implications of this on both METATEM execution and temporal resolution. Many restrictions mean that the logic becomes uninteresting as either eventualities cannot be constrained or the restricted logic reduces to classical logic. However, the particular restrictions defined below still allow temporal resolution to take place and non-trivial METATEM programs to be defined.

Let the following be the unrestricted form of the always rules

$$\left(\bigwedge_{j=1}^{r} \neg a_j \wedge \bigwedge_{k=1}^{s} b_k\right) \Rightarrow \bigcirc \left(\bigvee_{l=1}^{t} c_l \vee \bigvee_{m=1}^{u} \neg d_m\right)$$

where the a_j, b_k, c_l and d_m are all propositions. Similarly, we consider the unrestricted form of initial rules to be

$$\mathbf{start} \Rightarrow \bigvee_{l=1}^{t} c_l \vee \bigvee_{m=1}^{u} \neg d_m$$

and of sometime rules to be

$$\left(\bigwedge_{j=1}^{r} \neg a_j \wedge \bigwedge_{k=1}^{s} b_k\right) \Rightarrow \Diamond l$$

Now we restrict this general form in the following two ways.

R1: $r = 0$ and $t \leq 1$;
R2: $u \leq 1$ and $s = 0$.

Restriction R1 has always rules with conjunctions of propositions on the left-hand side and Horn clauses enclosed within a next operator on the right-hand side and initial rules with Horn clauses on the right hand side. Restriction R2 has always rules with conjunctions of negated propositions on the left-hand side and clauses with at most one negative literal enclosed within a next operator on the right-hand side. The initial rules similarly have clauses with at most one negative literal on the right-hand side.

In order that the resolution rules may be applied we define R1$^+$ and R2$^+$ restrictions that further restrict the sometime rules.

Definition 1. A set of SNF rules, R, is said to be R1$^+$ restricted if, and only if, $r \in R$ is of $R1$ format and if r is a sometime rule then the eventuality is a negated proposition.

Sets of R2$^+$ restricted SNF rules can be similarly defined. Although the results presented throughout the rest of this paper mainly concern the R1 restriction, dual results apply to R2 restricted rules.

2.4 Translation into RSNF

While any arbitrary PTL formula can be translated into a set of SNF rules, the use of RSNF (R1) introduces some restrictions on this. In particular, while the majority of PTL formulae can still be represented, formulae containing (non-conjunctive) binary operators in the scope of an even number of negations, where each of the operands is a positive literal, i.e.. '$a \lor b$', '$a \mathcal{W} b$', or '$a \mathcal{U} b$', where a and b are both propositions, can not. Note that this also includes formulae such as '$(\neg a \land \neg b) \Rightarrow \ldots$'.

3 Complexity

In this section we will show that the problem R1-SAT, of deciding the satisfiability of finite sets of R1 restricted formulae is just as hard as deciding satisfiability of general PTL formulae: we will show that it is PSPACE-complete in the total length of the rules.

Recall (from [SC85]) that satisfiability in the propositional temporal language with just $\bigcirc$ (called **X** by Sistla and Clarke) and $\Diamond$ (called **F**) is PSPACE-complete. It follows that R1-SAT is in PSPACE as any R1 restricted set of rules almost directly represents a formula of almost the same length in Sistla and Clarke's language.

To see that R1-SAT is PSPACE-hard, we will directly code the operation of a PSPACE Turing Machine M by a R1 restricted set of rules. We assume that M is space bounded by some polynomial $S(n)$ in n and $a = a_1 a_2 ... a_n$ is an input to M.

Due to space restrictions we omit full details of the coding but attempt to give an adequate sketch. The idea is to present a set of rules (of size polynomial in n) which is satisfiable, if at all, in a structure which corresponds closely to a successful run of M (if it has one).

This structure will use its first $S(n)$ states to code up the initial contents of M's tape, and the initial state of M. Then there will be a marker state. Then the next length of $S(n)$ states correspond to the contents of M's tape in the second moment of its successful run. This is also followed by a marker state. This pattern of lengths of $S(n)$ states separated by marker states continues indefinitely. The state corresponding to the tape square under the head at each stage will contain a proposition indicating the state in which M is at that stage. To code up this information we need propositions for the symbols in the alphabet and the states in M.

So far, the coding of the Turing machine in a propositional logic should be very familiar–see [SC85]. However, we have an unusually restricted language. To describe the structure using R1 rules is mostly straight forward using a few extra propositions. For example, we need propositions with meaning "the proposition for tape symbol e will be true in k states" for each e and each $k \leq S(n)$. We need rules which count down k. We also need many rules which say that two different tape symbols are not true at the same state. To enforce that there is only one machine state proposition true in each $S(n)$ length we use many rules containing propositions with meaning "the proposition for machine state q will be true in k states".

The rules which say how to change the tape from one step to the next are easy to state. However, due to the restrictions on use of negations it is harder to specify that tape symbols stay the same on other parts of the tape. To do this we introduce propositions δ_z where $-S(n) \leq z \leq S(n)$. The intuitive meaning of δ_z being true at the jth state in a certain length of $S(n)$ states corresponding to the ith step of M, is that at the ith step the head of M is over the $(j - z)$th square on the tape. We also make a new proposition R or L (but not both) true throughout a length of states if the machine moves right or left at that step. To do this, we must check, at a marker state, what machine state proposition lies ahead and which tape symbol lies the same distance ahead. A rule for each entry in the transition table of M will then give the truth of L or R. This can be perpetuated through the length. With the combination of R or L and δ_z it is then easy to tell if a tape symbol will be modified or not.

The rules for setting up the initial state of the machine and tape are straight forward given a. Finally we need a rule which says that the accepting state eventually occurs. For this we need a new proposition which holds at all structure states except where the accepting state proposition holds. It can be shown that the satisfiability of the resulting set of rules is equivalent to the acceptance of a by M. Thus R1-SAT is PSPACE-hard.

4 Applications of SNF

There are two main applications of SNF that have been developed. The first is the direct execution of temporal formulae in SNF, the second is a temporal resolution method that can be applied to formulae in SNF. Both of these make use of the succinct representation provided by the normal form. Before considering the effect of our restrictions on these applications, we will first outline these execution and resolution mechanisms.

4.1 METATEM

The idea behind METATEM [BFG+95, BFG+96] is to directly use the formula to be executed in order to build a model (in our case, a sequence) for the formula. Rather than providing a detailed description of the METATEM execution mechanism, we will show how SNF can be executed using the basic ideas from METATEM.

If a program, $\mathcal{P}$, is given as a set of SNF rules $\{\Phi_i \Rightarrow \Psi_i \mid 1 \leq i \leq n\}$, then this represents the PTL formula

$$\varphi_{\mathcal{P}} = \Box \bigwedge_{i=1}^{n} (\Phi_i \Rightarrow \Psi_i)\,.$$

The execution of the program, $\mathcal{P}$, is an iterative process of labelling a model structure with the propositions true in each state, which eventually yields a model for the formula, $\varphi_{\mathcal{P}}$ (if the formula is satisfiable). Essentially, the execution follows a forward chaining approach through the SNF rules. Choice during execution comes either from executing disjunctions of sometime rules. In the latter case, we can express this choice by the equivalence:

$$\Diamond a \Leftrightarrow (a \vee \bigcirc\Diamond a)$$

So to make $\Diamond a$ true, the execution mechanism can either make a true immediately, or, make $\Diamond a$ true in the next state. This latter step is done by passing a commitment $\Diamond a$ from the current state to the next state to be conjoined with the consequents of the successful rules. Thus, the execution mechanism has a strategy not only for choosing between disjuncts, but also for choosing when to satisfy formulae of the form $\Diamond a$ (called *eventualities*). In METATEM, the execution mechanism attempts to satisfy as many eventualities as possible at any moment. In the case of conflicting eventualities, e.g., $\Diamond a$ and $\Diamond\neg a$, the oldest outstanding eventuality is satisfied first. Any unsatisfied eventualities are passed on to the next state.

Backtracking If a contradiction is generated within a state, i.e., execution of the rules has forced us to make both a proposition and its negation true in the current state, then a form of 'backtracking' — undoing previous choices — occurs. This backtracking undoes the model construction and returns the execution to a previous choice point. If there are no more choices left, execution fails, signifying that the program is unsatisfiable.

Loop Checking An important aspect of METATEM, and one that ensures that the execution mechanism is complete for PTL, is the inclusion of *loop checking*. When the system is executing, it is possible for it to end up in a 'loop'. This is where the same sequence of states is recurring along with the same unsatisfied eventualities. Before constructing a new state, a check for such a situation is made. If this is detected, the execution is forced to backtrack to a previous choice point. Using this loop checking mechanism, it can be shown that the METATEM execution mechanism, when it uses the following strategy for choosing which disjunct to execute, is sound and complete with respect to the semantics of PTL. The choice strategy used is to try to execute disjuncts that will satisfy the longest outstanding eventualities first. Thus, an attempt to satisfy each individual eventuality will occur as the execution proceeds. For a longer description of the correctness of propositional METATEM, see [BFG+95].

Applications Although much of the development of METATEM has been suspended in favour of Concurrent METATEM [Fis94], the language has applications in system modelling [FFO93], databases [FMO91] and meta-level representation [BFGH91].

4.2 Temporal Resolution

As we also consider the effect of the various restrictions on SNF has on Fisher's temporal resolution method [Fis91], here we briefly review this method. The clausal temporal resolution method consists of repeated applications of both 'step' and 'temporal' resolution on sets of formulae in SNF, together with various simplification steps.

Step Resolution 'Step' resolution consists of the application of standard classical resolution rule to formulae representing constraints at a particular moment in time, together with simplification rules for transferring contradictions within states to constraints on previous states. Simplification and subsumption rules are also applied. Pairs of initial rules or always rules may be resolved using the following (step resolution) rules.

$$\frac{\begin{array}{l}\textbf{start} \Rightarrow A \lor r \\ \textbf{start} \Rightarrow B \lor \neg r\end{array}}{\textbf{start} \Rightarrow A \lor B} \qquad \frac{\begin{array}{r}P \Rightarrow \bigcirc(A \lor r) \\ Q \Rightarrow \bigcirc(B \lor \neg r)\end{array}}{(P \land Q) \Rightarrow \bigcirc(A \lor B)}$$

Once a contradiction within a state is found using step resolution, the following rule can be used to generate extra global constraints.

$$\frac{P \Rightarrow \bigcirc\textbf{false}}{\begin{array}{r}\textbf{start} \Rightarrow \neg P \\ \textbf{true} \Rightarrow \bigcirc\neg P\end{array}}$$

The step resolution process terminates when either no new resolvents are derived, or **false** is derived in the form of the rules **start** $\Rightarrow$ **false** or **true** $\Rightarrow \bigcirc$**false**.

Temporal Resolution The aim of temporal resolution is to resolve a sometime rule, $Q \Rightarrow \Diamond l$ with a set of rules that together imply $\Box\neg l$, for example a set of rules that together have the effect of $A \Rightarrow \bigcirc\Box\neg l$. However the interaction between the '$\bigcirc$' and '$\Box$' operators in PTL makes the definition of such a rule non-trivial and further the translation from PTL to SNF will have removed all but the outer level of $\Box$–operators. So, resolution will be between a sometime rule and a *set* of rules that together imply a $\Box$–formula contradicting the sometime rule. Thus, given a set of rules in SNF, then for every rule of the form $Q \Rightarrow \Diamond l$ temporal resolution may be applied between this sometime rule and a set of always rules, which taken together force $\neg l$ always to be satisfied. The temporal resolution rule is given by the following

$$\frac{\begin{array}{l}A_0 \Rightarrow \bigcirc F_0 \\ \ldots \quad \ldots \\ A_n \Rightarrow \bigcirc F_n \\ Q \Rightarrow \Diamond l\end{array}}{Q \Rightarrow (\bigwedge_{i=0}^{n} \neg A_i)\,\mathcal{W}\,l} \qquad \begin{array}{c}\text{with} \\ \text{side} \\ \text{conditions}\end{array} \quad \left\{\begin{array}{l}\text{for all } 0 \leq i \leq n \\ \vdash F_i \Rightarrow \neg l \\ \vdash F_i \Rightarrow \bigvee_{j=0}^{n} A_j\end{array}\right\}$$

where the side conditions ensure that the set of rules $A_i \Rightarrow \bigcirc F_i$ together imply $\Box\neg l$. So if any of the A_i are satisfied then $\neg l$ will be *always* be satisfied, i.e.,

$$\bigvee_{i=0}^{n} A_k \Rightarrow \bigcirc\Box\neg l.$$

Such a set of rules are known as a *loop* in $\neg l$. We note that the loops described here are like those we must check for in METATEM. However, in temporal resolution we are trying to *detect* loops to allow us to apply the temporal resolution rule and hence derive a contradiction, whilst in the execution of METATEM programs we are trying to *avoid* loops in order to generate consistent execution sequences.

Completeness of the resolution procedure has been shown in [Pei94].

Loop Search Different approaches to detecting loops, i.e. a set of rules that together imply $\Box\neg l$, have been investigated in [Dix96]. Here, we describe one of these approaches, namely a Breadth-First Search Algorithm. The graph constructed using this approach is a sequence of nodes that are labelled with formulae in DNF. This represents the left-hand sides of rules used to expand the previous node which have been disjoined and simplified.

Breadth-First Search Algorithm For each rule of the form $Q \Rightarrow \Diamond l$ carry out the following.

1. Search for all the rules of the form $X_k \Rightarrow \bigcirc\neg l$, for $k = 0$ to b, disjoin the left-hand sides and make the *top node* H_0 equivalent to this, i.e.

$$H_0 \Leftrightarrow \bigvee_{k=0}^{b} X_k.$$

   Simplify H_0. If $\vdash H_0 \Leftrightarrow$ **true** we terminate having found a loop.
2. Given node H_i, build node H_{i+1} for $i = 0, 1, \ldots$ by looking for rules or combinations of rules of the form $A_j \Rightarrow \bigcirc B_j$, for $j = 0$ to m where $\vdash B_j \Rightarrow H_i$ and $\vdash A_j \Rightarrow H_0$. Disjoin the left-hand sides so that

$$H_{i+1} \Leftrightarrow \bigvee_{j=0}^{m} A_j$$

   and simplify as previously.
3. Repeat (2) until
   (a) $\vdash H_i \Leftrightarrow$ **true**. We terminate having found a Breadth-First loop and return **true**.
   (b) $\vdash H_i \Leftrightarrow H_{i+1}$. We terminate having found a Breadth-First loop and return the DNF formula H_i.
   (c) The new node is empty. We terminate without having found a loop.

5 The Effect of Restrictions on METATEM

While, in §4, we presented an outline of both temporal execution and resolution, in §5 and §6 respectively, we consider the effect of restricting SNF on these two applications.

We begin with the effect on expressiveness within METATEM.

5.1 Expressiveness

In this section we show that for any temporal formula in the full propositional language there is a set of RSNF rules such that running METATEM on the rules will produce a model of the original formula. This shows there is no difference in absolute expressive power between the restricted syntax and full propositional temporal logic in terms of METATEM programs.

Suppose that ϕ is a formula from PTL using the propositions from P. Using the algorithms of Vardi and Wolper [VW86] or Emerson and Sistla [ES84] we can build a nondeterministic finite state Büchi 2^P-automaton $A = (S, T, S_0, F)$ which recognizes a P-structure if, and only if, it is a model of ϕ.

We could determinize A using results of McNaughton [McN66] or Safra [SV89] and then translate the deterministic (Rabin) automaton into a METATEM program. Instead, as we are just interested in describing one model of ϕ, we can do this by direct examination of the transition table T of A. We find a loop of states of A which can be reached from some state in S_0 and which contain at least one element from the acceptance set F. Say that an *accepting accessed loop* in A is a pair

$$\begin{array}{l}(((r_0, b_0, r_1), (r_1, b_1, r_2), ..., (r_{m-1}, b_{m-1}, s_0)),\\ ((s_0, a_0, s_1), (s_1, a_1, s_2), ..., (s_{n-1}, a_{n-1}, s_0)))\end{array}$$

of sequences of triples from T such that:

- $r_0 \in S_0$
- all the $r_0, ..., r_{m-1}, s_0, ..., s_{n-1}$ are distinct
- some s_i is in F

It is clear that there is an algorithm to find any accepting accessed loop in A if there is one and report not if there is not. It is also clear that ϕ has no models if A has no accepting accessed loop. If A has such a loop as above then we can use it to construct a model of ϕ and an RSNF program which constructs the same model. Simply define $l : \mathbb{N} \longrightarrow 2^P$ by $l(0) = b_0, l(1) = b_1, ..., l(m-1) = b_{m-1}, l(m) = a_0, l(m+1) = a_1, ..., l(m+n-1) = a_{n-1}, l(m+n) = a_0, l(m+n+1) = a_1, ..., l(m+2n) = a_0$, etc.

The chosen model of ϕ is that in which $p \in P$ is true at time i if, and only if, $p \in l(i)$. To construct this model we can use the following RSNF METATEM program which uses propositions from P as well as some new ones $B_0, B_1, ..., B_{m-1}, A_0, A_1, ..., A_{n-1}$.

The program contains rules:

- **start** $\Rightarrow B_0$
- $B_i \Rightarrow \bigcirc B_{i+1}$ for $i = 0, ..., m-2$
- $B_{m-1} \Rightarrow \bigcirc A_0$
- $A_i \Rightarrow \bigcirc A_{i+1}$ for $i = 0, ..., n-2$
- $A_{n-1} \Rightarrow \bigcirc A_0$
- $A_i \Rightarrow \bigcirc p$ for all $p \in a_{i+1}$, for all $i = 0, ..., n-2$,
- $A_{n-1} \Rightarrow \bigcirc p$ for all $p \in a_0$
- **start** $\Rightarrow p$ for all $p \in b_0$
- $B_i \Rightarrow \bigcirc p$ for all $p \in b_{i+1}$, for $i = 0, ..., m-2$
- $B_{n-1} \Rightarrow \bigcirc p$ for all $p \in a_0$

Provided that this program is run with a version of METATEM where atoms default to false unless otherwise constrained, this program will construct the chosen model of ϕ.

5.2 Eliminating Backtracking

Using similar automata-theoretic techniques to §5.1 we can show how to eliminate nondeterminism from RSNF programs. Here, we show how, given a set R of RSNF rules, we can construct a set R' of RSNF rules such that R' is a deterministic program and yet it constructs a model of R. In fact, as R directly represents a temporal logic formula we could use the techniques of the previous subsection to accomplish this task. However, we present an alternative algorithm (also exponentially complex), because it proceeds via the interesting step of demonstrating how to represent the METATEM program R as a nondeterministic finite state Büchi automaton with a trivial acceptance condition —an object commonly known as a transducer. If R uses propositional atoms from P, the transducer's language will be 2^P.

First choose new atoms e_c for each proposition c which appears in the head of a sometime rule in R. Let E be the set of such atoms. The set S of states of the transducer will be those $s \in 2^{P \cup E}$ such that:

- if $(\bigwedge_{i \in I} p_i \Rightarrow \Diamond c) \in R$ and each $p_i \in s$ then either $c \in s$ or $e_c \in s$.

We define $A = (S, T, S_0, F)$, with $F = true$ as follows. Let S_0 contain all the subsets s of $P \cup E$ which themselves contain at least one proposition from the head of each initial always rule in R. The transition table $T \subseteq S \times 2^P \times S$ is given by $(s, a, r) \in T$ if, and only if,:

- $a = P \cap s$
- if $(\bigwedge_{i \in I} p_i \Rightarrow \bigcirc \bigvee_{j \in J} \neg q_j) \in R$ and each $p_i \in s$ then there is some $j \in J$ such that $q_j \notin t$;
- if $(\bigwedge_{i \in I} p_i \Rightarrow \bigcirc (\bigvee_{j \in J} \neg q_j \vee q)) \in R$ each $p_i \in s$ and each $q_j \in t$ then $q \in t$;
- if $e_c \in s$ and $c \notin s$ then $e_c \in t$.

It should be clear that an accepting run σ of A corresponds exactly to a possible model of R provided that whenever e_c is an element of a state σ_i then there is some $j \geq i$ such that $c \in \sigma_j$. Using the same reasoning as in the last subsection we deduce that R has a model if, and only if, there is a pair

$$(((s_0, b_0, s_1), (s_1, b_1, s_2), ..., (s_{m-1}, b_{m-1}, r_0)),$$
$$((r_0, a_0, r_1), (r_1, a_1, r_2), ..., (r_{n-1}, a_{n-1}, r_0)))$$

of sequences of triples from T such that:

- $s_0 \in S_0$
- all the $s_0, ..., s_{m-1}, r_0, ..., r_{n-1}$ are distinct
- if $e_c \in r_0 \cup r_1 \cup ... \cup r_{n-1}$ then $c \in r_0 \cup r_1 \cup ... \cup r_{n-1}$.

We can also turn this sequence of states into a new deterministic RSNF program R' by using new propositions to represent the states which appear. This program will produce a model of R. The details are similar to those in the last subsection and we omit them.

5.3 Efficiency

As a typical METATEM program is designed not to terminate, usual measures of computational complexity are irrelevant. However, for the task of comparing different versions of METATEM, such as the restricted syntax version considered here, it is interesting to examine some measure of how long it takes for the execution process to complete one cycle from deciding on the truths of propositions at one state to those at the next state in the structure it is constructing. Note that any such measure ignores complications due to backtracking. In what follows we shall also ignore the process of loop checking which, for the sake of completeness of the process, should be occurring at each step but which, as we have seen earlier, may be a time consuming process.

So what we consider here is how many computation steps does it take to get from the state of truth of propositions at time k to the state of truth of propositions at time $k+1$ when executing an RNSF program, R. The input to this one-cycle processor will be the program R, the truth values for all the propositions and truth values for any of the eventualities mentioned in the program. This latter set of values will be needed to keep track of which eventualities must be satisfied at some future stage. The output of the process may be **false** if the structure can not be extended by one step, or a list of new truth values for propositions and eventualities.

Let us suppose that the size of R is n. This also bounds the number of propositions and the number of eventualities. Here we show that the one-cycle problem is PTIME computable — in fact cubic in n. This contrasts with general METATEM programs where the problem is NP-complete.

Checking which program rules are "fired" is at most quadratic — we only need check the truth of a conjunction of propositions. Next, we only have at most n non-temporal Horn clauses to solve. This is also quadratic as we may have n rounds of checking the clauses to reach the fixpoint solution. If there is no solution then the structure can not be extended and we are finished. Next we check which eventualities can be satisfied. This is best done by choosing the oldest unsatisfied eventuality (age can be indicated by the order of listing of eventualities in the input), and seeing whether its proposition can be made true consistently with the propositions already determined as being true. Again this involves n rounds of checking clauses. Whether or not this eventuality can be satisfied at this stage, it is also best to move on to check the second oldest eventuality and so on. This is where the process becomes cubic. Finally we need check which eventualities have to be passed on to the $k+1$st step. This is just the union of those left unsatisfied from the last step, with those whose rules are "fired" at this step. Overall, this is a process of cubic complexity.

5.4 "Programmability"

Note, while the transformations given in §5.1 and §5.2 showed that RSNF has the same expressive power as SNF, each of these required an exponential increase in the number of rules. Here we (briefly) consider how useful RSNF is in its own right, i.e. what types of problems can be naturally represented using sets of RSNF rules. A typical R1 rule is of the form

$$(a_1 \wedge a_2 \wedge \ldots \wedge a_n) \Rightarrow \bigcirc((b_1 \wedge b_2 \wedge \ldots \wedge b_m) \Rightarrow c)$$

where all of a_i, b_j and c are propositions. In order to fire such a rule, the a_i conjunction of propositions must occur. Once fired, this provides a Horn clause to be computed in the next state. Thus, sets of such RSNF rules can be seen as defining which sets of Horn clauses will be used in each state. In this sense, the execution of RSNF rules can be seen as an intuitive generalisation of standard Horn clause execution.

With respect to the types of program that can be coded in RSNF, the rule form presented above covers many programming clichés seen in METATEM. In particular, the forms that are disallowed, i.e. basing current action on a conjunction of negative literals in the previous state, and describing choices such as $a \vee b$ or $a \mathcal{U} b$ are rare. However, it is likely that a practical implementation of RSNF execution will allow both R1 and R2 restricted rules (see §7).

6 The Effect of Restrictions on Resolution

First we show that by restricting the sometime rules all the resolvents produced can be rewritten as RSNF rules.

Theorem 2. *Given a set of rules, R, that are R1$^+$ restricted then the application of any resolution rule produces resolvents that are R1$^+$ restricted.*

Proof. We must consider the application of both step and temporal resolution rules. First we consider step resolution. The left hand sides of rules remain in the correct format as we either have two initial rules with **start** on their left hand sides or we have two always rules both with conjunctions of propositions (or **true**) on their left hand sides. Applying the step resolution rule to either two initial rules or two always rules, the resolution of two Horn clauses on the right hand side produces a Horn clause, **true** or **false**. Rules with **true** on the right hand side are removed through simplification as they always hold. The generation of **start** $\Rightarrow$ **false** or **true** $\Rightarrow \bigcirc$**false** means that we have detected a contradiction. The generation of $P \Rightarrow \bigcirc$**false** from the resolution of two always rules means that the rule must be rewritten as **start** $\Rightarrow \neg P$ and **true** $\Rightarrow \bigcirc \neg P$. Again these fit the rule structure as P was a conjunction of propositions (if P was **true** we would have terminated at the previous step) so $\neg P$ is a disjunction of negated propositions and is therefore a Horn clause.

When applying the temporal resolution rule we search for a set of rules $A_i \Rightarrow \bigcirc B_i$ for $i = 0 \dots n$ that satisfy the conditions for being a loop given in §4.2. Each of these rules is made from combining several always rules where each left hand side is a conjunction of propositions. Assume the sometime rule is $Q \Rightarrow \Diamond \neg l$ where from the R1$^+$ restriction Q is a conjunction of propositions and l is a proposition. The resolvent in this case will be

$$Q \Rightarrow (\bigwedge_{i=0}^{n} \neg A_i)\, \mathcal{W}\, \neg l.$$

When rewriting to the normal form we obtain the rules

$$\begin{array}{rlrl} \textbf{start} & \Rightarrow \neg Q \vee \neg l \vee t & \textbf{start} & \Rightarrow \neg Q \vee \neg l \vee \neg A_i \\ \textbf{true} & \Rightarrow \bigcirc(\neg Q \vee \neg l \vee t) & \textbf{true} & \Rightarrow \bigcirc(\neg Q \vee \neg l \vee \neg A_i) \\ t & \Rightarrow \bigcirc(\neg l \vee t) & t & \Rightarrow \bigcirc(\neg l \vee \neg A_i) \end{array}$$

for each A_i where $i = 0 \dots n$ and where t is a fresh proposition. So as each A_i and Q are conjunctions of propositions the rules are still in the R1$^+$ rule format.

Our first observation is related to loops that can be detected in the R1$^+$ restriction.

Lemma 3. *For any loop, $A_i \Rightarrow \bigcirc B_i$ for $i = 0 \dots n$, in the proposition l (where B_i cannot be equivalent to* **false***) formed from merging rules in the R1 format we can find a loop $A'_i \Rightarrow \bigcirc B'_i$ such that $A_i \Rightarrow A'_i$ for each i and B'_i is a conjunction of propositions.*

Proof. From the definition of a loop in l, $A_i \Rightarrow \bigcirc B_i$, $\vdash B_i \Rightarrow \bigvee_{j=0}^{n} A_j$ and $\vdash B_i \Rightarrow l$ for each i. From the R1 restriction all the left hand sides of the always rules are conjunctions of propositions therefore $\bigvee_{i=0}^{n} A_j$ is a disjunction of conjunctions of propositions. Thus B_i must be a conjunction of propositions or we can delete one or more of the rules that has been combined to make $A_i \Rightarrow \bigcirc B_i$ for a particular i and still have a loop. As we have only deleted rules from the rules that make up each merged rule then each $A_i \Rightarrow A_i'$ and B_i' is a conjunction of propositions as required.

Lemma 4. *For any loop in l, $A_i \Rightarrow \bigcirc B_i$, a loop in l, $A_i' \Rightarrow \bigcirc B_i'$, can be constructed such that $A_i \Rightarrow A_i'$ by merging only rules of the form $X \Rightarrow \bigcirc p$ where X is a conjunction of propositions and p is a proposition.*

Proof. From Lemma 3 given any loop $A_i \Rightarrow \bigcirc B_i$ we can construct a (possibly more general) loop, $A_i' \Rightarrow \bigcirc B_i'$, such that $A_i \Rightarrow A_i'$ for each i and B_i' is a conjunction of propositions. Either each B_i' has been made up from combining rules with a single proposition on its right hand side and we are done or a set or rules, R, have been merged that contain a rule r with two or more disjuncts on the right hand side. As B_i' is a conjunction of propositions it means that by performing step resolution (and subsumption) we may resolve all such rules with other rules in R until R contains only rules of the required form.

Step resolution is now more efficient because we have Horn clauses on the right hand side of rules so strategies such as unit resolution may be applied. Unit resolution is complete for classical propositional Horn formulae [WCR64], and algorithms have been developed that test for satisfiability of classical propositional Horn formulae based on unit resolution in polynomial time [JL77] . Thus although we must repeatedly perform classical style resolution proofs on the right hand side of rules (because when $\bigcirc$**false** is detected the rule is rewritten) the resolution steps to derive each $\bigcirc$**false** on the right hand sides of rules will be more efficient as less intermediate steps need to be carried out. So step resolution will be more efficient.

Loop search is also more efficient as loops consist of sets of merged rules with conjunctions of propositions on the left and right hand sides. Carrying out Breadth-First Search will be more efficient for the following reasons.

1. From Lemmas 3 and 4 we know that for any loop in l we can find a more general loop that is constructed from merging rules with single propositions on the right hand side. Assuming all unit resolution possible has been carried out, in most cases the number of rules that loop search is applied to is reduced as we can ignore any always rules that do not fit into this class. To apply the Breadth-First Search algorithm we must find merged rules that satisfy certain criteria. In the worst case we must check all possible combinations of (always) rules, i.e. for m rules there are 2^m merged rules. Thus we reduce m. In particular, given n propositions there are 2^n possible left hand sides of rules in the required format. Similarly there are $2^n - 1$ different right hand sides that are disjunctions of negated propositions and $n \times 2^{n-1}$ with exactly one proposition and zero or more negated propositions. Therefore given n propositions we may have of the order of $n2^{2n-1}$ rules. As we have the R1 restriction we need only consider rules that have a single proposition on the right hand side, i.e. $n \times 2^n$ rules.
2. Given a Breadth-First Search node H_j to expand using a merged rule $A_i \Rightarrow \bigcirc B_i$ we must ensure that $\vdash B_i \Rightarrow H_j$. In the unrestricted form both these formulae are in DNF. So for n propositions we would need to check 2^n lines in a truth table to ensure validity. Using the R1 restriction B_i is a conjunction of propositions so to check the implication hold we only need to build the lines in the truth table where all the literals in B_i are set to true. Thus in the worst case where B_i is a single literal we only need to check half as many lines, i.e. 2^{n-1} lines.

3. The final governing factor in Breadth-First Search is the number of nodes that must be constructed. For unrestricted Breadth-First search the maximum will be 2^{n-1}. This can be shown by construction a (*behaviour*) graph whose nodes are valuations of the set of propositions. Paths through the graph represent all possible models of the set of always rules. The set of nodes containing the required literal, l is identified and nodes with any edges leading out of this set deleted. Nodes that lead out of the new set are again deleted until the set is empty or no more nodes can be deleted. The latter represents a loop in l and each cycle of deletions corresponds to the construction of a node in the Breadth-First algorithm. Thus the worst case is when the behaviour graph constructed is a sequence of nodes and we need 2^{n-1} deletion cycles for n propositions. For the R1 restriction this must be reduced as it is not possible to construct such a graph.

Other loop search algorithms have been given in [Dix96]. One of these, Depth-First Search becomes feasible in this setting. The Depth-First Search algorithm is similar to Breadth-First Search but instead of using all the rules to expand the graph we use them one at a time. The problem is that each rule we use with two or more disjuncts on the right hand side represents branching in the graph constructed and we must check that each branch leads back into the set of nodes already constructed. As we know loops are constructed from merged rules with conjunctions of propositions on the left and right hand sides we avoid the extra search required to check each branch.

7 Conclusions

In this paper we have considered particular restrictions of the SNF normal form related to Horn clauses (and their duals). While the overall complexity of the decision problem for the logic remains the same (see §3), we have shown that both execution and resolution can be implemented more efficiently in this restricted version. In particular, the complexity of executing one step of METATEM reduces from NP-complete to PTIME, as does the complexity of deriving $P \Rightarrow \bigcirc$**false** in step resolution, while the Breadth-First Search algorithm used in temporal resolution produces improved efficiency at each stage. Thus, our future work concerns the implementation and testing of both execution and resolution, together with further analysis of similar restrictions. Finally, in practical systems, it is likely to be the case that both R1 and R2 restricted rules would be utilised, but that separate mechanisms would be employed on each rule type.

Acknowledgements This work was partially supported by EPSRC under Research Grant GR/K57282 and GR/K54946.

References

[Aba87] M. Abadi. *Temporal-Logic Theorem Proving*. PhD thesis, Department of Computer Science, Stanford University, March 1987.

[BFG+95] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: An Introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

[BFG+96] 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.

[BFGH91] H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. Meta-Reasoning in Executable Temporal Logic. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, April 1991.

[Cha70] C. L. Chang. The Unit Proof and the Input Proof in Theorem Proving. *ACM Journal*, 17:698–707, 1970.

[dCP92] L. Fariñas del Cerro and M. Penttonen, editors. *Intensional Logics for Programming*. Oxford University Press, 1992.

[Dix96] C. Dixon. Search Strategies for Resolution in Temporal Logics. In *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE)*, volume 1104 of *Lecture Notes in Computer Science*, New Brunswick, New Jersey, July/August 1996.

[ES84] E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61:175–201, 1984.

[FFO93] M. Finger, M. Fisher, and R. Owens. 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.

[Fis91] M. Fisher. A Resolution Method for Temporal Logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[Fis94] M. Fisher. A Survey of Concurrent METATEM — The Language and its Applications. In *First International Conference on Temporal Logic (ICTL)*, Bonn, Germany, July 1994.

[Fis97] M. Fisher. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4), August 1997.

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

[FO95] M. Fisher and R. Owens, editors. *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, February 1995.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proceedings of the Seventh Symposium on the Principles of Programming Languages*, 1980.

[JL77] N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3:107–117, 1977.

[McN66] R. McNaughton. Testing and Generating Infinite Sequences by a Finite Automaton. *Information and Control*, 9:521–530, 1966.

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

[Pei94] M. Peim. Propositional Temporal Resolution Over Labelled Transition Systems. Unpublished Technical Note, Department of Computer Science, University of Manchester, 1994.

[Rob65] J. A. Robinson. A Machine–Oriented Logic Based on the Resolution Principle. *ACM Journal*, 12(1):23–41, January 1965.

[SC85] A. P. Sistla and E. M. Clarke. Complexity of propositional linear temporal logics. *ACM Journal*, 32(3):733–749, July 1985.

[SV89] S. Safra and M. Y. Vardi. On ω-Automata and Temporal Logic. In *STOC*, pages 127–137, Seattle, Washington, May 1989. ACM.

[VW86] M. Y. Vardi and P. Wolper. Automata-theoretic Techniques for Modal Logics of Programs. *Journal of Computer and System Sciences*, 32(2):183–219, April 1986.

[WCR64] L. Wos, D. Carson, and G. Robinson. The Unit Preference Strategy in Theorem Proving. In *Proceedings of AFIPS Fall Joint Computer Conference*. Thompson Book Co., 1964.

[Wol85] P. Wolper. The Tableau Method for Temporal Logic: An overview. *Logique et Analyse*, 110–111:119–136, June-Sept 1985.

This article was processed using the LaTeX macro package with LLNCS style