A Normal Form for Temporal Logics and its Applications in Theorem-Proving and Execution

Michael Fisher

Department of Computing
Manchester Metropolitan University
Manchester M1 5GD, U.K.

EMAIL: M.Fisher@doc.mmu.ac.uk

Abstract

In this paper a normal form, called *Separated Normal Form (SNF)*, for temporal logic formulae is described. A simple propositional temporal logic, based on a discrete linear model structure, is introduced and a procedure for transforming an arbitrary formula of this logic into SNF is described. It is shown that the transformation process preserves satisfiability and ensures that any model of the transformed formula is a model of the original one. This normal form not only provides a simple and concise representation for temporal formulae, but is also used as the basis for both a resolution proof method and an execution mechanism for this type of temporal logic. In addition to outlining these applications, we show how the normal form can be extended to deal with first-order temporal logic.

1 Introduction

This paper provides a description of *Separated Normal Form (SNF)*, a normal form for temporal logic formulae. This normal form provides a simple and concise representation for temporal statements, with formulae being represented solely in terms of three core temporal operators. In addition to these representational benefits, SNF provides the basis for a proof method for discrete temporal logic based upon clausal resolution [18]. As SNF formulae comprise the key elements required for temporal descriptions of dynamic systems, namely information about the current state of the system, what the system can do *next* and what it must do at *some time* in the future, it is not surprising that SNF is also used as the basis of an execution mechanism for temporal logics. This language, called METATEM [4, 17], has been used to represent and animate temporal formulae describing a range of dynamic, concurrent and distributed systems. The practical implementations of both the temporal resolution method and the METATEM family of languages make extensive use of the transformation into SNF of the form described in this paper [17, 10]. This paper provides a refined and extended account of SNF, the transformation of arbitrary formulae to SNF, and the applications of SNF. As such, it collects together, and extends, work from a number of previous papers.

In order to present SNF we introduce a simple propositional temporal logic, based on a

linear discrete model of time, in §2. Both the normal form and the procedure for transforming arbitrary formulae of the logic into SNF are described and analysed in §3. In §4, the principal applications of SNF are outlined, while in §5 the normal form is extended to a first-order temporal logic. Finally, in §6, both a summary and an indication of future work are presented.

2 Propositional Temporal Logic

This section describes a standard propositional temporal logic [11], called PTL, based on a discrete, linear model of time, with a finite past and infinite future [22]. This temporal logic can be seen as classical logic extended with various modalities, for example '$\Diamond$', '$\Box$', and '$\bigcirc$'. The intuitive meaning of these connectives is as follows: $\Diamond A$ is true now if A is true *sometime* in the future; $\Box A$ is true now if A is true *always* in the future; and $\bigcirc A$ is true now if A is true at the *next* moment in time. In this presentation, similar connectives are introduced to enable reasoning about the *past* [26].

2.1 Syntax

We begin with the formal syntax of the language. Formulae of PTL are constructed using the following symbols.

- A set, $\mathcal{L}_p$, of *propositional symbols* represented by strings of lower-case alphabetic characters.
- Classical connectives, $\neg$, $\vee$, $\wedge$, **true**, **false** and $\Rightarrow$.
- Future-time temporal operators, categorised as
  - unary operators: $\bigcirc$, $\Diamond$, $\Box$,
  - binary operators: $\mathcal{U}$, and $\mathcal{W}$.
- Past-time temporal operators, categorised as
  - unary operators: $\bigcirc\!\!\!\!\bullet$, $\bullet$, $\blacklozenge$, $\blacksquare$,
  - binary operators: $\mathcal{S}$, and $\mathcal{Z}$.
- '(' and ')' which are, as usual, used to avoid ambiguity.

The set of *well-formed formulae* of PTL (WFF_p) is defined as follows.

- Any element of $\mathcal{L}_p$ is in WFF_p.
- If A and B are in WFF_p, then so are

$\neg A$	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	(A)	
$\Diamond A$	$\Box A$	$A \mathcal{U} B$	$A \mathcal{W} B$	$\bigcirc A$	
$\blacklozenge A$	$\blacksquare A$	$A \mathcal{S} B$	$A \mathcal{Z} B$	$\bullet A$	$\bigcirc\!\!\!\!\bullet A$

Formulae in WFF_p can be classified as follows. A *literal* is either a proposition (i.e., an element of $\mathcal{L}_p$), or the negation of a proposition. A *state-formula* is either a literal or a boolean combination of other state-formulae. *Future-time formulae* (non-strict) are defined as follows.

- If A is a state-formula, then A is a future-time formula.
- If A and B are future-time formulae, then $\neg A$, $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \mathcal{U} B$, $A \mathcal{W} B$, $\bigcirc A$, $\Diamond A$, and $\Box B$ are all future-time formulae.

Strict past-time formulae are defined as follows.

- If A and B are either state-formulae or strict past-time formulae, then $\circledcirc A$, $\bullet B$, $A \mathcal{S} B$, $A \mathcal{Z} B$, $\blacklozenge A$, and $\blacksquare B$ are all strict past-time formulae.
- If A and B are strict past-time formulae, then $\neg A$, $A \wedge B$, $A \vee B$, and $A \Rightarrow B$ are all strict past-time formulae.

2.2 Semantics

Intuitively, the models for PTL formulae are based on discrete, linear structures having a finite past and infinite future, i.e., sequences such as

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

where each s_i, called a *state*, provides a propositional valuation. However, rather than representing the model structure in this way, we will define a model, σ, as

$$\sigma = \langle \mathbf{N}, \pi_p \rangle$$

where

$\mathbf{N}$ is the Natural Numbers, which is used to represent the sequence $s_0, s_1, s_2, s_3, \ldots$, and,

π_p is a map from $\mathbf{N} \times \mathcal{L}_p$ to $\{\mathsf{T}, \mathsf{F}\}$, giving a propositional valuation for each state in the sequence.

An interpretation for this logic is defined as a pair $\langle \sigma, i \rangle$, where σ is the model and i the index of the state at which the temporal statement is to be interpreted.

A semantics for well-formed temporal formulae is a relation between interpretations and formulae, and is defined inductively as follows, with the (infix) semantic relation being represented by '$\models$'. The semantics of a proposition is defined by the valuation given to that proposition at a particular state:

$$\langle \sigma, i \rangle \models p \quad \text{iff} \quad \pi_p(i, p) = \mathsf{T} \qquad [\text{for } p \in \mathcal{L}_p].$$

The semantics of the standard propositional connectives is as in classical logic, e.g.,

$$\langle \sigma, i \rangle \models A \vee B \quad \text{iff} \quad \langle \sigma, i \rangle \models A \quad \text{or} \quad \langle \sigma, i \rangle \models B.$$

The semantics of the unary future-time temporal operators is defined as follows.

$$\begin{array}{lll} \langle \sigma, i \rangle \models \bigcirc A & \text{iff} & \langle \sigma, i+1 \rangle \models A \\ \langle \sigma, i \rangle \models \Diamond A & \text{iff} & \text{there exists } j \in \mathbf{N} \text{ such that } j \geq i \text{ and } \langle \sigma, j \rangle \models A \\ \langle \sigma, i \rangle \models \Box A & \text{iff} & \text{for all } j \in \mathbf{N}, \text{ if } j \geq i \text{ then } \langle \sigma, j \rangle \models A \end{array}$$

The informal semantics of these operators are as follows: $\bigcirc A$ means that A must be satisfied in the *next* state; $\Diamond A$ means that A must be satisfied at *some* state in the future; $\Box A$ means that A must be satisfied at *all* states in the future. Additionally, the syntax includes two binary future-time temporal operators, interpreted as follows.

$$\langle \sigma, i \rangle \models A \mathcal{U} B \quad \text{iff} \quad \begin{array}{l} \text{there exists } k \in \mathbf{N}, \text{ such that } k \geq i \text{ and } \langle \sigma, k \rangle \models B \text{ and} \\ \quad \text{for all } j \in \mathbf{N}, \text{ if } i \leq j < k \text{ then } \langle \sigma, j \rangle \models A \end{array}$$

$$\langle \sigma, i \rangle \models A \mathcal{W} B \quad \text{iff} \quad \langle \sigma, i \rangle \models A \mathcal{U} B \quad \text{or} \quad \langle \sigma, i \rangle \models \Box A$$

Both $A \mathcal{U} B$ and $A \mathcal{W} B$ mean that A must be satisfied up until the point in time where B is satisfied. The difference between the two operators is that $A \mathcal{U} B$ implies that B *must* be satisfied at some point in the future, while $A \mathcal{W} B$ does not. This gives the following equivalence.

$$(A \mathcal{W} B \wedge \Diamond B) \quad \Leftrightarrow \quad A \mathcal{U} B$$

As temporal formulae are interpreted at a particular state-index, i, then indices less than i represent states that are 'in the past' with respect to state s_i. The semantics of the unary past-time operators is given as follows.

$$\begin{array}{lll} \langle \sigma, i \rangle \models \bullet A & \text{iff} & \langle \sigma, i-1 \rangle \models A \quad \text{or} \quad i = 0 \\ \langle \sigma, i \rangle \models \circledcirc A & \text{iff} & \langle \sigma, i-1 \rangle \models A \quad \text{and} \quad i > 0 \\ \langle \sigma, i \rangle \models \blacklozenge A & \text{iff} & \text{there exists } j \in \mathbf{N}, \text{ such that } 0 \leq j < i \text{ and } \langle \sigma, j \rangle \models A \\ \langle \sigma, i \rangle \models \blacksquare A & \text{iff} & \text{for all } j \in \mathbf{N}, \text{ if } 0 \leq j < i \text{ then } \langle \sigma, j \rangle \models A \end{array}$$

Note that, in contrast to the future-time operators, the '$\blacklozenge$' ("sometime in the past") and '$\blacksquare$' ("always in the past") operators are interpreted as being *strict*, i.e., the current index is not included in their definition. Also, as there is a unique start state, termed the *beginning of time*, two different last-time operators are used. The difference between the '$\circledcirc$' and '$\bullet$' operators is that for any formula A, $\bullet A$ is satisfied, while $\circledcirc A$ is not, when interpreted at the beginning of time. In particular, $\bullet$**false** is *only* satisfied when interpreted at the beginning of time. Note that the following equivalence relates these two last-time operators.

$$\circledcirc \neg A \;\Leftrightarrow\; \neg \bullet A$$

Apart from their strictness, the binary past-time operators are similar to their future-time counterparts; their semantics is defined as follows.

$$\langle \sigma, i \rangle \models A \mathcal{S} B \quad \text{iff} \quad \begin{array}{l} \text{there exists } k \in \mathbf{N}, \text{ such that } 0 \leq k < i \text{ and } \langle \sigma, k \rangle \models B \text{ and} \\ \quad \text{for all } j \in \mathbf{N}, \text{ if } k < j < i \text{ then } \langle \sigma, j \rangle \models A \end{array}$$

$$\langle \sigma, i \rangle \models A \mathcal{Z} B \quad \text{iff} \quad \langle \sigma, i \rangle \models A \mathcal{S} B \quad \text{or} \quad \langle \sigma, i \rangle \models \blacksquare A.$$

The $\Diamond$ and $\Box$ (and their past-time counterparts) can be derived from the $\mathcal{U}$ and $\mathcal{W}$ operators ($\mathcal{S}$ and $\mathcal{Z}$ respectively) as follows:

$$\begin{array}{lll} \Diamond A & \Leftrightarrow & \textbf{true} \, \mathcal{U} A \\ \Box A & \Leftrightarrow & A \mathcal{W} \textbf{false} \end{array}$$

We now give the definitions of the satisfaction of formulae by models that will be used later.

Definition 1 (Satisfaction) A well-formed formula, A, is **satisfied** in a particular model, σ, at a state-index, i, if, and only if, $\langle \sigma, i \rangle \models A$. Similarly, a well-formed formula, A, is **satisfied in a model**, σ, if, and only if, A is satisfied in σ, at state-index 0.

Definition 2 (Satisfiability) A well-formed formula, A, is **satisfiable** if, and only if, there exists a model in which A is satisfied.

Thus, a formula, A, is satisfiable if, and only if, '$\exists\sigma.\ \langle\sigma, 0\rangle \models A$'. An alternative definition of satisfiability, sometimes found in the literature, is that a formula, A, is satisfiable if, and only if, '$\exists\sigma.\ \exists i \in \mathbf{N}.\ \langle\sigma, i\rangle \models A$'. Note that satisfiability in terms of either of these definitions can be transformed into satisfiability in terms of the other.

3 A Normal Form for Propositional Temporal Logic

The normal form that we introduce is called Separated Normal Form [18]. It was originally derived as part of the development of an executable subset of temporal logic [4, 16], but was found to have wider applicability, particularly in temporal theorem-proving. The variety we introduce here is a slight generalisation of that found in our earlier work, but can easily be transformed if necessary.

3.1 Separated Normal Form

Formulae in Separated Normal Form (SNF) are of the form

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

Here, each P_i is a *non-strict*[1] past-time temporal formula and each F_i is a *non-strict* future-time formula. Each of the '$P_i \Rightarrow F_i$' (called *rules*) is further restricted to be of one the following

$$\bullet\mathbf{false} \quad \Rightarrow \quad \bigvee_{c=1}^{n} r_c \qquad \text{(an } \textit{initial} \text{ rule)}$$

$$\bigwedge_{a=1}^{l} p_a \wedge \bigcirc \bigwedge_{b=1}^{m} q_b \quad \Rightarrow \quad \bigvee_{c=1}^{n} r_c \qquad \text{(a } \Box\text{-rule)}$$

$$\bigwedge_{a=1}^{l} p_a \wedge \bigcirc \bigwedge_{b=1}^{m} q_b \quad \Rightarrow \quad \Diamond s \qquad \text{(a } \Diamond\text{-rule)}$$

where each p_a, q_b, r_c or s is a literal.

Recall that the formula $\bullet$**false** can *only* be satisfied at the beginning of time, thus ensuring that the initial rule can only be applied there, while any formula of the form $\bigcirc A$ can *never* be satisfied at the beginning of time, thus ensuring that the $\Box$ and $\Diamond$-rules can be applied everywhere.

In this simple propositional case, the conjunction of literals on the left-hand side of each $\Box$-rule, i.e.

$$\bigwedge_{a=1}^{l} p_a$$

can be moved on to the right-hand side by negating each conjunction. For example,

$$\bigwedge_{a=1}^{l} p_a \wedge \bigcirc \bigwedge_{b=1}^{m} q_b \Rightarrow \bigvee_{c=1}^{n} r_c$$

[1] Here, 'non-strict' means "including the present".

becomes

$$\bigcirc \bigwedge_{b=1}^{m} q_b \;\Rightarrow\; \bigvee_{a=1}^{l} \neg p_a \vee \bigvee_{c=1}^{n} r_c \,.$$

Such a transformation is used, for example, in the execution mechanism for temporal formulae (see §4.2) where the left-hand sides of each rule simply represent 'firing conditions' from the previous state. Note, however, that in the first-order case (see §5) the above transformation may not be desirable.

Benefits of Concise Representation

An important observation about SNF is that a wide range of temporal attributes can be represented using only a simple set of operators. These operators allow us to describe properties of the current state, of the transitions that can occur between the current and the *next* state, and of situations that will occur at some, unspecified, state in the future. This feature has implications for the use of temporal logic as a programming language (see §4.2), where a range of powerful temporal constructs can be coded in terms of these simple operators. Hence, the programming language implementor need only efficiently implement the basic temporal operations, while programmers have a simple, yet powerful, description language at their disposal.

3.2 Translating to SNF

In this section we describe an algorithm for transforming an arbitrary PTL formula into a set of SNF rules (comprising purely past-time formulae on their left-hand sides, as described above). Rather than justifying the correctness of the translation here, we will simply describe the main steps, returning to the discussion of correctness in §3.3.

Before providing the detail of the algorithm we will review the technique of *renaming* which is used extensively in this algorithm.

Renaming Transformations

Renaming was originally used in classical logic to preserve the structure of formulae when rewriting to a normal form [29, 9]. The technique simply consists of replacing a sub-formula by a new proposition symbol and linking the truth value of this new symbol to the sub-formula that has been replaced. In PTL, we must also ensure that this link is maintained at every moment in time.

Consider, as an example, the temporal formula $\Diamond(a \wedge (b \mathcal{U} c))$. If the subformula '$b \mathcal{U} c$' is renamed using the new proposition symbol 'x', the full formula becomes

$$\Diamond(a \wedge x) \wedge \Box(x \Leftrightarrow (b \mathcal{U} c)) \,.$$

Note that, we can replace the '$\Leftrightarrow$' in the additional renamed formula by '$\Rightarrow$', only under the condition that the renamed subformula has positive polarity (i.e., occurs under an even number of negations). Although such simplifications will be utilised later we will not apply them in the PTL to SNF translation described here. We also note that though using '$\Rightarrow$' instead of

'$\Leftrightarrow$' preserves satisfiability, a model for the transformed set of rules is not guaranteed to be a model for the original formula[2].

Although we have described this renaming as an operation on PTL formulae, it technically represents a translation of formulae in Quantified Propositional Temporal Logic (QPTL), an extension of PTL in which quantification over propositions is allowed [33]. Thus, the new formula is really

$$\exists x.\ \Diamond(a \wedge x) \wedge \Box(x \Leftrightarrow (b\,\mathcal{U}\,c))\,.$$

However, as we only use QPTL in order to introduce new propositions, e.g. x, and as we do not require that the new formula be logically equivalent to the original one, we can ignore this aspect until the correctness of this renaming procedure is discussed (in §3.3).

We now describe the procedure for translating an arbitrary PTL formula into SNF.

From PTL to NNF$_p$

The first step in the transformation simply involves rewriting the formula so that all negations only appear when applied to propositions. This is analogous to translating classical formulae into Negation Normal Form (NNF), and utilises the obvious correspondences:

$$\begin{array}{rcl@{\qquad}rcl}
\neg(A \wedge B) & \rightarrow & \neg A \vee \neg B & \neg(A \vee B) & \rightarrow & \neg A \wedge \neg B \\
\neg\mathbf{true} & \rightarrow & \mathbf{false} & \neg\mathbf{false} & \rightarrow & \mathbf{true} \\
\neg\neg A & \rightarrow & A & \neg \circledcirc A & \rightarrow & \bullet \neg A \\
\neg \bigcirc A & \rightarrow & \bigcirc \neg A & \neg \bullet A & \rightarrow & \circledcirc \neg A \\
\neg \Box A & \rightarrow & \Diamond \neg A & \neg \blacksquare A & \rightarrow & \blacklozenge \neg A \\
\neg \Diamond A & \rightarrow & \Box \neg A & \neg \blacklozenge A & \rightarrow & \blacksquare \neg A \\
\neg(A\,\mathcal{U}\,B) & \rightarrow & (\neg B)\,\mathcal{W}\,(\neg A \wedge \neg B) & \neg(A\,\mathcal{S}\,B) & \rightarrow & (\neg B)\,\mathcal{Z}\,(\neg A \wedge \neg B) \\
\neg(A\,\mathcal{W}\,B) & \rightarrow & (\neg B)\,\mathcal{U}\,(\neg A \wedge \neg B) & \neg(A\,\mathcal{Z}\,B) & \rightarrow & (\neg B)\,\mathcal{S}\,(\neg A \wedge \neg B)
\end{array}$$

These are applied exhaustively to the formula, thus producing a new formula in Negation Normal Form for PTL (NNF$_p$).

From NNF$_p$ to Flat Rule Form

A formula in *Rule Form* is simply written as

$$\Box \bigwedge_i (P_i \Rightarrow F_i)$$

where each P_i is a (non-strict) past-time formula and each F_i is a (non-strict) future-time formula. *Flat Rule Form* is a further refinement of this where neither P_i nor F_i contain nested temporal operators and where F_i contains at most one temporal operator.

Translating from a general NNF$_p$ formula to Flat Rule Form consists of three distinct steps, as follows.

1. Apply

$$A \longrightarrow \Box(\bullet\,\mathbf{false} \Rightarrow A)$$

[2]This follows from the fact that if $X \Rightarrow Y$, then *models(X)* $\subseteq$ *models(Y)* and so some models of Y might not be models of X.

which ensures that the formula is of the correct general structure (i.e. $\Box(X \Rightarrow Y)$). This transformation can be justified by recalling that for A to be satisfiable, some model must be found such that A is satisfied by that model at the beginning of time. Thus, anchoring the formula to the beginning of time (as above) has no effect on its satisfiability.

2. Use renaming to ensure that the components of each $P \Rightarrow F$ are in the appropriate *past* $\Rightarrow$ *future* form. Traditionally, this requires *separation* [24, 23], however we can utilise renaming to achieve the same effect. For example, to extract a past-time temporal formula, Q, from inside a formula containing the '$\mathcal{U}$' operator, we can use the following transformation rule[3].

$$\{P \Rightarrow A \mathcal{U} Q\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} P & \Rightarrow & A \mathcal{U} x \\ Q & \Leftrightarrow & x \end{array} \right\}$$

3. Again use renaming, but this time to ensure that no nested temporal operators occur. For example, we might rename the rule $P \Rightarrow \Diamond(R \mathcal{U} S)$ as follows.

$$\{P \Rightarrow \Diamond(R \mathcal{U} S)\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} P & \Rightarrow & \Diamond y \\ y & \Leftrightarrow & R \mathcal{U} S \end{array} \right\}$$

Again, both 'x' and 'y' are a *new* proposition symbols.

In both steps (2) and (3) above we use renaming transformations of the general form

$$\{P \Rightarrow \mathcal{F}(\mathcal{A})\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} P & \Rightarrow & \mathcal{F}(z) \\ \mathcal{A} & \Leftrightarrow & z \end{array} \right\}$$

where $\mathcal{F}(\mathcal{A})$ is a temporal formula containing the sub-formula $\mathcal{A}$.

Removing Unwanted Temporal Operators

We now turn to the removal of temporal operators from rules in flat rule form. We will present a series of transformation rules which assume that the operator to be removed appears on its own on either the left-hand or right-hand side of a rule. Before describing these transformations, we will review how to ensure that temporal operators appear on their own at one side of a rule.

Recall that, in flat rule form, not only are there no nested temporal operators, but also there is at most one temporal operator appearing on the right-hand side of each rule. As all negation operators only apply to propositions, then the only operators, beside the principal temporal operator, that can appear on either side of a rule are '$\vee$' and '$\wedge$'. In order to ensure that one side of a rule contains a formula with the temporal operator at its top level we can apply a variety of rewrite rules.

The first pair of rewrite rules can be used to ensure that the top-level operator of the left-hand side of a rule is temporal.

$$\{P \vee Q \Rightarrow F\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} P & \Rightarrow & F \\ Q & \Rightarrow & F \end{array} \right\} \qquad \text{(where } P, Q \text{ are arbitrary formulae)}$$

$$\{P \wedge Q \Rightarrow F\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} P & \Rightarrow & \neg Q \vee F \end{array} \right\} \qquad \text{(where } Q \text{ is a non-temporal formula)}$$

[3] We introduce rules of the form '$A \Leftrightarrow B$' for clarity, though these can later be reduced to a pair of rules such as '$A \Rightarrow B$' and '$\neg A \Rightarrow \neg B$'. These new rules must, in turn, be translated into NNF_p.

The second pair of rewrite rules carry out a similar process, but for the right-hand side of rules.

$$\{P \Rightarrow F \wedge G\} \longrightarrow \left\{ \begin{array}{lcl} P & \Rightarrow & F \\ P & \Rightarrow & G \end{array} \right\} \qquad \text{(where } F, G \text{ are arbitrary formulae)}$$

$$\{P \Rightarrow F \vee G\} \longrightarrow \left\{ \begin{array}{lcl} P \wedge \neg G & \Rightarrow & F \end{array} \right\} \qquad \text{(where } G \text{ is a non-temporal formula)}$$

Obviously, we must ensure that the second rules in each pair are not applied continuously as each could undo the action of the other.

We can now proceed to remove the unwanted future-time temporal operators from the rules in flat rule form, as follows.

Removal of $\Box$: $\{P \Rightarrow \Box A\} \longrightarrow \left\{ \begin{array}{lcl} P & \Rightarrow & A \wedge y \\ \bigcirc y & \Leftrightarrow & A \wedge y \end{array} \right\}$

Removal of $\bigcirc$: $\{P \Rightarrow \bigcirc A\} \longrightarrow \left\{ \begin{array}{lcl} P & \Rightarrow & z \\ \bigcirc z & \Leftrightarrow & A \end{array} \right\}$

Removal of $\mathcal{W}$: $\{P \Rightarrow A \mathcal{W} B\} \longrightarrow \left\{ \begin{array}{lcl} P & \Rightarrow & B \vee (A \wedge w) \\ \bigcirc w & \Leftrightarrow & B \vee (A \wedge w) \end{array} \right\}$

Removal of $\mathcal{U}$: $\{P \Rightarrow A \mathcal{U} B\} \longrightarrow \left\{ \begin{array}{lcl} P & \Rightarrow & B \vee (A \wedge w) \\ P & \Rightarrow & \Diamond B \\ \bigcirc w & \Leftrightarrow & B \vee (A \wedge w) \end{array} \right\}$

Note that the '$\mathcal{U}$' operator is transformed into a set of rules including the '$\Diamond$' operator. This is related to the fact that '$\mathcal{U}$' can be defined as a minimal fixpoint [33]. The transformation rules for the past-time operators are derived in a similar way. For example, the rules for the '$\mathcal{S}$' (a minimal fixpoint), and '$\mathcal{Z}$' (a maximal fixpoint) are as follows. However, note that because of the finite past constraint, the '$\blacklozenge$' operator is not required in the translation of '$\mathcal{S}$'.

Removal of $\mathcal{S}$: $\{A \mathcal{S} B \Rightarrow F\} \longrightarrow \left\{ \begin{array}{lcl} \bigcirc(B \vee (A \wedge x)) & \Rightarrow & F \\ \bigcirc(B \vee (A \wedge x)) & \Leftrightarrow & x \end{array} \right\}$

Removal of $\mathcal{Z}$: $\{A \mathcal{Z} B \Rightarrow F\} \longrightarrow \left\{ \begin{array}{lcl} \bullet(B \vee (A \wedge w)) & \Rightarrow & F \\ \bullet(B \vee (A \wedge w)) & \Leftrightarrow & w \end{array} \right\}$

Note that in all these transformations removing temporal operators, the symbols *w*, *x*, *y* and *z* again represent *new* proposition symbols.

Into SNF

By this stage, we have reduced the temporal operators to those required for SNF, and have a set of rules in the correct general form. The final step to SNF is to manipulate the classical operators on each side of these rules to ensure exactly the SNF form. During this process, we rewrite the right-hand side of each rule into Conjunctive Normal Form (CNF), treating formulae of the form $\Diamond l$ as literals, and rewrite the left-hand side of each rule into Disjunctive

Normal Form (DNF) in the usual way, but with the following additional rewrite rules.

$$\begin{array}{rcl}
\bullet P \wedge \bullet Q & \longrightarrow & \bullet(P \wedge Q) \\
\circledcirc P \wedge \bullet Q & \longrightarrow & \circledcirc(P \wedge Q) \\
\bullet P \wedge \circledcirc Q & \longrightarrow & \circledcirc(P \wedge Q) \\
\circledcirc P \wedge \circledcirc Q & \longrightarrow & \circledcirc(P \wedge Q) \\
\circledcirc(P \vee Q) & \longrightarrow & \circledcirc P \vee \circledcirc Q \\
\bullet(P \vee Q) & \longrightarrow & \bullet P \vee \bullet Q
\end{array}$$

Finally, the following rewrites are exhaustively applied to each rule.

$$\begin{array}{rcl}
\{P \Rightarrow (F \wedge G)\} & \longrightarrow & \left\{ \begin{array}{lcl} P & \Rightarrow & F \\ P & \Rightarrow & G \end{array} \right\} \\
\{(P \vee Q) \Rightarrow F\} & \longrightarrow & \left\{ \begin{array}{lcl} P & \Rightarrow & F \\ Q & \Rightarrow & F \end{array} \right\} \\
\{\bullet R \Rightarrow F\} & \longrightarrow & \left\{ \begin{array}{lcl} \bullet \mathbf{false} & \Rightarrow & F \\ \circledcirc R & \Rightarrow & F \end{array} \right\} \\
\{P \Rightarrow \bigvee_i F_i \vee \Diamond l\} & \longrightarrow & \{\bigwedge_i \neg F_i \wedge P \Rightarrow \Diamond l\}
\end{array}$$

The first two rules simply de-construct conjunctive formulae, the third rule ensures that the '$\bullet$' operator only applies to '**false**', while the final rule ensures that $\Diamond$-rules contain nothing but the $\Diamond$-formula on their right-hand side.

3.3 Correctness of PTL to SNF Translation Steps

We will now provide correctness results for the translation from PTL to SNF presented in §3.2. As mentioned above, if we are working in the framework of QPTL the translation process results in a formula that is logically equivalent to the original formula. Thus, assuming that the transformation procedure is characterised by 'τ', where for any formula A, $\tau(A)$ is in SNF, then we would expect that

$$\vdash_{\text{QPTL}} A \Leftrightarrow \exists \bar{x}.\, \tau(A)$$

where $\bar{x}$ is the set of new propositions introduced during the translation process.

However, as we are considering a translation within PTL, rather than QPTL, the two properties that we prove about the translation process are as follows.

Theorem 3 (Preservation of Satisfiability) Given a formula A in WFF_p, if A is satisfiable, so is $\tau(A)$.

Theorem 4 (Reliability of Models) Given a formula A in WFF_p, if $\tau(A)$ is satisfied in a model σ, then A is also satisfied in σ.

If we use renaming that introduces new rules of the form '$x \Rightarrow A$', rather than '$x \Leftrightarrow A$', Theorem 3 still holds, but Theorem 4 does not. Though both results are desirable, for example if we wish to execute the transformed formula (see §4.2), certain applications, such as temporal resolution, only require Theorem 3 (see §4.1).

Before considering the full PTL to SNF translation process, we will prove that the basic renaming procedure for PTL formulae has the above properties (assuming '$\Leftrightarrow$' is used in the renaming). In order to do this for the general case, we first introduce the concept of a *context*.

Definition 5 (Context) A *context* is taken to be a function over WFF_p which embeds its argument at a given position in an enclosing formula.

For example, we might define a context $\mathcal{P}$ as follows.

$$\mathcal{P}(X) \equiv \Diamond(a \wedge X)$$

where X is any formula in WFF_p.

Given the above definition of a context, we can state the general theorem about renaming, as follows.

Theorem 6 (Renaming) Given a formula, $\mathcal{A}(B)$, where $\mathcal{A}$ is a context and B is in WFF_p, then let C be the formula

$$\mathcal{A}(b) \wedge \Box(b \Leftrightarrow B)$$

where 'b' is a new propositional symbol (i.e. b does not occur anywhere within $\mathcal{A}$ or B). Given these constraints,

1. if $\mathcal{A}(B)$ is satisfiable, then C is satisfiable, i.e.

   $$\text{if} \quad \exists\sigma.\ \langle\sigma,0\rangle \models \mathcal{A}(B) \quad \text{then} \quad \exists\sigma'.\ \langle\sigma',0\rangle \models C,$$

   and,

2. any model for C is a model for $\mathcal{A}(B)$, i.e.

   $$\forall\sigma.\ \text{if} \quad \langle\sigma,0\rangle \models C \quad \text{then} \quad \langle\sigma,0\rangle \models \mathcal{A}(B).$$

Proof Assuming a particular formula $\mathcal{A}(B)$, in which the subformula B is to be renamed, we prove the two parts of this theorem separately as follows.

1. *if $\mathcal{A}(B)$ is satisfiable, then $\mathcal{A}(b) \wedge \Box(b \Leftrightarrow B)$ is satisfiable.*

   Prove by contradiction. Assume that $\mathcal{A}(B)$ is satisfiable and, in particular, that it is satisfied by the model σ, i.e. $\langle\sigma,0\rangle \models \mathcal{A}(B)$. Now also assume that $\mathcal{A}(b) \wedge \Box(b \Leftrightarrow B)$ is unsatisfiable. Given these assumptions, we can derive a contradiction as follows.

   Given that $\langle\sigma,0\rangle \models \mathcal{A}(B)$, then either

   (a) the satisfaction of the formula $\mathcal{A}(B)$ within σ does not depend upon B at any time, or,

   (b) the satisfaction of the formula $\mathcal{A}(B)$ within σ depends on the satisfaction of B at at least one point in σ.

   In the first case, we can replace B by any proposition, so long as that proposition does not already occur in $\mathcal{A}$, and satisfaction in σ will be preserved. In particular, we can replace B by the new proposition symbol b and we know that $\langle\sigma,0\rangle \models \mathcal{A}(b)$, thus giving a contradiction.

   In the second case, by the semantics of PTL, whatever form the context $\mathcal{A}$ takes, the question of the satisfiability of $\mathcal{A}(B)$ within σ will depend upon the satisfiability of B within σ, i.e., whether $\langle\sigma,i\rangle \models B$ for a particular index i. Recall that the proposition b is defined in such a way that it is satisfied at a particular moment in time if, and only

if, B is satisfied at that point. Thus, since $\langle \sigma, i \rangle \models B$ then we can construct a model σ' that agrees with σ except that $\langle \sigma', i \rangle \models b$ if, and only if, $\langle \sigma, i \rangle \models B$. Now, since b is a new proposition symbol, then we know that $\langle \sigma', 0 \rangle \models \mathcal{A}(b)$

Thus, $\langle \sigma', 0 \rangle \models \mathcal{A}(b) \wedge \Box(b \Leftrightarrow B)$, giving a contradiction.

2. *if* $\langle \sigma, 0 \rangle \models \mathcal{A}(b) \wedge \Box(b \Leftrightarrow B)$, *then* $\langle \sigma, 0 \rangle \models \mathcal{A}(B)$.

   As $\langle \sigma, 0 \rangle \models \mathcal{A}(b) \wedge \Box(b \Leftrightarrow B)$ then, within σ, the proposition b is satisfied at exactly those indices that the formula B is satisfied at. In particular, wherever the formula $\mathcal{A}(b)$ depends on the satisfaction of b, then B will be satisfied at that point. Thus, whatever the context $\mathcal{A}$ is, then σ is also a model for $\mathcal{A}(B)$.

   Note again that, since the 'new' symbol b does not occur at all within $\mathcal{A}(B)$, then its value does not affect whether $\mathcal{A}(B)$ is satisfied in any model or not.

Removal of Maximal Temporal Fixpoints

The other major transformation that we utilise in the PTL to SNF translation is that of replacing various (future-time) temporal operators by their fixpoint definitions[4]. This is based upon the simulation of fixpoints using QPTL [33], and particularly concerns the removal of those temporal operators represented as *maximal* fixpoints, i.e., $\Box$ and $\mathcal{W}$. Note that the $\mathcal{U}$ operator can be represented as a combination of operators based upon maximal fixpoints and the $\Diamond$ operator (which is retained within SNF), i.e.

$$P \mathcal{U} Q \equiv P \mathcal{W} Q \wedge \Diamond Q$$

Recall that a variety of temporal operators can be represented as fixpoints, e.g.

$$P \mathcal{W} Q \equiv \nu\xi.\ Q \vee (P \wedge \bigcirc \xi)$$

and we use the representation of such fixpoints in QPTL [33, 3] as the basis for our transformations. Note that, while the fixpoint representation can also be used for past-time formulae, each fixpoint formula will only have at most one solution (due to the finiteness of the past). Consequently, there is no distinction between maximal and minimal fixpoints in this past-time case. We can state the properties we require of such transformations more formally, as follows.

Theorem 7 (Fixpoint Removal) Consider a future-time temporal operator corresponding to a maximal fixpoint, say $\nu\xi.\ F(\xi)$. If the operator occurs in a formula within a temporal context, for example '$\mathcal{C}$', then occurrences of the operator can be replaced by a new proposition 'r', with the new formula $\Box(r \Leftrightarrow F(r))$ conjoined to the whole formula. This transformation satisfies

1. if $\mathcal{C}(\nu\xi.\ F(\xi))$ is satisfiable, then $\mathcal{C}(r) \wedge \Box(r \Leftrightarrow F(r))$ is satisfiable, and,
2. any model for $\mathcal{C}(r) \wedge \Box(r \Leftrightarrow F(r))$ is a model for $\mathcal{C}(\nu\xi.\ F(\xi))$.

[4]The removal of past-time operators is achieved in a different way, associated with the finite-past restriction.

Proof A fixpoint of the form

$$\nu\xi.\ F(\xi)$$

can be represented by the following QPTL formula [33].

$$\exists x.\ \underbrace{x \Leftrightarrow F(x)}_{\text{Existence}} \wedge \underbrace{\forall y.\ \Box(y \Leftrightarrow F(y)) \ \Rightarrow\ (y \Rightarrow x)}_{\text{Maximality}} \wedge x$$

The 'existence' part of this formula states that there is some proposition, x, which satisfies the fixpoint formula $x \Leftrightarrow F(x)$, while the 'maximality' part ensures that this particular proposition is the greatest solution of the fixpoint[5].

Now, recall that replacing the propositional variable 'x' in $\exists x.\ x \Leftrightarrow F(x)$ by a new proposition, 'p' (i.e., one that has not been used before) is an operation that preserves our two required properties. This gives us

$$p \Leftrightarrow F(p)$$

together with the appropriate minimality/maximality constraint. However, if we apply this translation to maximal fixpoints, then we can remove the need to have an explicit maximality constraint by enclosing the above formula by a '$\Box$' operator. For example, as

$$P \mathcal{W} Q \ \equiv\ \nu\xi.\ Q \vee (P \wedge \bigcirc \xi)$$

then any occurrence of $P \mathcal{W} Q$ can be replaced by the (new) proposition 'u', with the following formula added.

$$\Box(u \Leftrightarrow Q \vee (P \wedge \bigcirc u)$$

Thus, the translation of maximal fixpoints into QPTL, the removal of existential quantification from the derived QPTL formula, and the removal of explicit maximality conditions all preserve the two desired properties.

Correctness of full PTL to SNF translation

We can now prove both Theorems 3 and 4. We do this by showing that each step of the transformation from PTL to SNF preserves the two properties required, namely

- if a formula A is satisfiable, then the transformed formula, $\tau(A)$, is also satisfiable, and,
- if the transformed formula, $\tau(A)$, has a model, then this is also a model of A.

In fact, the majority of the translation steps preserve logical equivalence which, in turn, implies both these properties. In such cases, we will omit the proofs.

We consider each step in the translation process in turn.

1. **From PTL to NNF$_p$**

   As this involves simply 'pushing' negation operators through the temporal formula, and as each particular rule applied can be justified directly from the semantics of PTL, this step produces a formula that is logically equivalent to the original.

2. **From NNF$_p$ to Flat Rule Form**

   This step consists of three particular transformations.

[5] Note that $x \sqsubseteq y$ if, and only if, $x \Rightarrow y$.

(a) $A \longrightarrow \Box(\bullet\,\mathbf{false} \Rightarrow A)$

Recall that we are only interested in the satisfaction of a formula at state 0. Thus, while the $\Box$ operator means the formula must be satisfied at *all* states, the '$\bullet\,\mathbf{false}\ \Rightarrow$' ensures that the formula is anchored at state 0. Consequently, a formula that is logically equivalent to the original is produced.

(b) Renaming for embedded past-time formulae.

Here, renaming is used and so, by Theorem 6, both the desired properties are preserved.

(c) Renaming for embedded future-time formulae.

Again, by Theorem 6, both the desired properties are preserved.

3. **Removing Unwanted Temporal Operators**

This step utilises the removal of two forms of temporal operator.

(a) Simple transformations, such as the removal of '$\bigcirc$':

$$\{P \Rightarrow \bigcirc A\} \longrightarrow \left\{ \begin{array}{lcl} P & \Rightarrow & z \\ \bullet z & \Leftrightarrow & A \end{array} \right\}$$

From the semantics of PTL, we can immediately see that the desired properties are preserved.

(b) In the case of future-time temporal operators, we rewrite $\mathcal{U}$ into a combination of $\mathcal{W}$ and $\Diamond$ operators and use Theorem 7 to show that removing maximal fixpoints preserves the required properties.

In the case of the past-time temporal operators $\mathcal{S}$ and $\mathcal{Z}$, we observe that because of the finite past employed in PTL, the maximal and minimal fixpoints coincide. In particular, by unwinding the fixpoint definitions a finite number of times, we can provide the value of either type of fixpoint, the only difference being their value at the beginning of time.

4. **Into SNF**

The rules applied in this step consist of classical rewrite rules such as those used to transform to CNF and DNF, together with simple temporal rewrites such as

$$\bullet P \wedge \bullet Q \longrightarrow \bullet(P \wedge Q).$$

All these transformations produce a formula that is logically equivalent to the original formula. The proof of this follows simply from the semantics of PTL.

Thus, we have shown that each of the transformation steps preserve the properties outlined in Theorems 3 and 4.

Relaxing Constraints

We briefly mention the effect of using transformations that introduce implications, rather than equivalences. We note that both the renaming procedure and the removal of maximal

fixpoints can be refined to introduce '$\Rightarrow$', rather than '$\Leftrightarrow$'. Although the set of SNF rules produced in this case is smaller than it would have been if '$\Leftrightarrow$' we used, the Renaming Theorem (Theorem 6) must be weakened. In particular, the second condition, namely that any model for the transformed formula is a model for the original formula, is lost. Thus the revised Renaming Theorem is as follows.

Theorem 8 (Renaming (Revised)) Given a formula, $\mathcal{A}(B)$, where $\mathcal{A}$ is a context, B is in WFF_p, and B occurs within an even number of negations within $\mathcal{A}$, then let C be the formula

$$\mathcal{A}(b) \wedge \Box(b \Rightarrow B)$$

where 'b' is a new propositional symbol (i.e. b does not occur anywhere within $\mathcal{A}$ or B). Given these constraints, if $\mathcal{A}(B)$ is satisfiable, then C is satisfiable, i.e.

$$\text{if} \quad \exists\sigma.\ \langle\sigma,0\rangle \models \mathcal{A}(B) \quad \text{then} \quad \exists\sigma'.\ \langle\sigma',0\rangle \models C.$$

3.4 Complexity of PTL to SNF Translation

To give an indication of the complexity of the transformation from PTL to SNF, we will consider the main components of the transformation process separately, as follows.

Complexity of Renaming Transformations

In classical propositional logic, the renaming of formulae to derive CNF is linear in the length of the formula [29]. In our transformation to SNF, a similar complexity is found for renaming transformations using '$\Rightarrow$', while exponential complexity occurs when '$\Leftrightarrow$' is used.

Renaming using '$\Leftrightarrow$' Assuming that no temporal transformations are required, the worst case is when each subformula of the initial formula must be renamed by a new proposition. In this situation, both the following transformations must be applied to rename all the subformulae of each component of a rule.

$$\{\ \mathcal{P}(A)\ \Rightarrow\ F\ \} \quad \longrightarrow \quad \left\{ \begin{array}{rcl} \mathcal{P}(x) & \Rightarrow & F \\ A & \Leftrightarrow & x \end{array} \right\}$$

$$\{\ P\ \Rightarrow\ \mathcal{F}(B)\ \} \quad \longrightarrow \quad \left\{ \begin{array}{rcl} P & \Rightarrow & \mathcal{F}(y) \\ y & \Leftrightarrow & B \end{array} \right\}$$

In each case, the renamed subformula will be duplicated, giving a potential exponential increase in the size of the formula. Note, however, that in practice the renamed formulae tend to be relatively simple.

Renaming using '$\Rightarrow$' The worst case, as above, is when each subformula of the initial formula must be renamed by a new proposition. Again, both the following transformations must be applied to rename all the subformulae of each component of a rule.

$$\{\ \mathcal{P}(A)\ \Rightarrow\ F\ \} \quad \longrightarrow \quad \left\{ \begin{array}{rcl} \mathcal{P}(x) & \Rightarrow & F \\ A & \Rightarrow & x \end{array} \right\}$$

$$\{\, P \Rightarrow \mathcal{F}(B) \,\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} P & \Rightarrow & \mathcal{F}(y) \\ y & \Rightarrow & B \end{array} \right\}$$

Here, however, the subformulae (A and B) are not duplicated and there is simply a linear increase in both the number of rules and the number of propositions.

Complexity of Transformations for Removing Temporal Operators

In a similar way to the renaming case above, several transformations that remove temporal operators not only add new rules, but also duplicate subformulae in these new rules. Again, if '$\Leftrightarrow$' is used, an exponential increase in the size of the formula is possible, while if '$\Rightarrow$' is used, we can show that only a linear increase occurs.

For example, the transformation that involves the largest duplication of subformulae is that for removing the '$\mathcal{Z}$' operator:

$$\{\, A \mathcal{Z} B \Rightarrow F \,\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} \bullet(B \vee (A \wedge w)) & \Rightarrow & F \\ \bullet(B \vee (A \wedge w)) & \Rightarrow & w \end{array} \right\}$$

Once the past-time component is rewritten, this new set of rules expands to six rules containing three occurrences the subformula F, together with two occurrences of both A and B. However, if we carry out three renaming transformations *before* this temporal transformation, we can ensure that the formulae A, B and F are replaced by new proposition symbols. Thus, renaming A, B and F by a, b and f respectively, the transformation of $A \mathcal{Z} B \Rightarrow F$ becomes:

$$\{A \mathcal{Z} B \Rightarrow F\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl} A & \Rightarrow & a \\ B & \Rightarrow & b \\ f & \Rightarrow & F \\ \circledcirc b & \Rightarrow & f \\ \circledcirc (a \wedge w) & \Rightarrow & f \\ \circledcirc b & \Rightarrow & w \\ \circledcirc (a \wedge w) & \Rightarrow & w \\ \bullet \mathbf{false} & \Rightarrow & f \\ \bullet \mathbf{false} & \Rightarrow & w \end{array} \right\}$$

Thus, in the worst case, temporal transformations of this form give a linear increase in the number of rules (though 9-fold in this case!) and a linear increase in the number of proposition symbols.

Complexity of Remaining Transformations

Of the remaining transformations, the one that has the largest potential complexity is that occurring in the final phase where the left-hand side of each rule is rewritten into DNF. Thus, in the worse case, this will be an exponential operation, though in most practical cases the formulae involved will be very small.

We have shown that the overall complexity of the transformation is exponential at worst, though in most practical situations the complexity will be governed by whether we use '$\Rightarrow$' or '$\Leftrightarrow$'. In particular, unless the DNF translation in the final part of the process dominates, the transformation using '$\Rightarrow$' will involve a linear increase in the number of rules and a linear increase in the number of proposition symbols.

3.5 Examples

In this section we present brief examples of the transformation from PTL to SNF.

Example 1

As an example of rewriting an arbitrary temporal formula into SNF, consider the following (contradictory) formula.

$$\Box(a \Rightarrow \bigcirc(l \wedge a)) \ \wedge \ a \ \wedge \ l \ \wedge \ \bigcirc\Diamond\neg l$$

Converting this formula to SNF, the following set of rules is generated.

1. $\bullet$**false** $\Rightarrow$ a
2. $\bullet$**false** $\Rightarrow$ l
3. $\bullet$**false** $\Rightarrow$ b
4. $\circledcirc a \Rightarrow l$
5. $\circledcirc a \Rightarrow a$
6. $\circledcirc b \Rightarrow \Diamond\neg l$

The first three rules represent the fact that a, l, and b must be satisfied at the first state. The fourth rule says that whenever a is satisfied, then l will be satisfied in the next state, and the fifth rule says that if a is satisfied, then a will also be satisfied in the next state. The final rule states that once b is satisfied, $\Diamond\neg l$ will be satisfied in the next state.

Note that, as these rules are applied globally, once a has been satisfied, rule 5 ensures that a will always be satisfied. In this case, rule 4 ensures that l will always be satisfied.

Example 2

Consider the temporal formula

$$p\,\mathcal{U}\,(a\,\mathcal{S}\,b) \wedge \Diamond\Box c\,.$$

The translation process can be applied to produce the following set of SNF rules.

1. $\bullet$**false** $\Rightarrow$ $d \vee f$
2. $\circledcirc f \Rightarrow \Diamond d$
3. $d \Rightarrow c$
4. $\circledcirc d \Rightarrow d$
5. $\bullet$**false** $\Rightarrow$ $e \vee p$
6. $\bullet$**false** $\Rightarrow$ $e \vee w$
7. $\bullet$**false** $\Rightarrow$ $e \vee g$
8. $\circledcirc g \Rightarrow \Diamond e$
9. $\circledcirc w \Rightarrow e \vee p$
10. $\circledcirc w \Rightarrow e \vee w$
11. $\circledcirc b \Rightarrow e$
12. $\circledcirc(a \wedge e) \Rightarrow e$

Here, rules 1–4 capture the formula '$\Diamond\Box c$', rules 4–10 capture '$p\,\mathcal{U}\,e$', and rules 11 and 12 define e to be '$a\,\mathcal{S}\,b$'.

4 Applications of SNF

The two main applications of SNF immediately follow from Theorems 3 and 4. The first is a temporal resolution method that can be applied to formulae in SNF, which relies on Theorem 3. The second is the direct execution of temporal formulae in SNF, which relies on both theorems.

4.1 Temporal Resolution

The classical resolution rule (for propositional logic) matches complementary literals in different subformulae [30], i.e.

$$\frac{A \vee p \qquad B \vee \neg p}{A \vee B}$$

The development of resolution has not only led to the development of fast theorem-provers for classical logics, but is also the basis for a variety of programming languages [25]. Indeed, proof procedures based on this rule are the most widely used method of mechanising classical logics [30, 35]. In spite of this, few resolution methods have been developed for temporal logics. The majority of theorem-proving tools for temporal logics have been based on either tableaux or automata [34, 31]. One resolution method that has been developed is based on *non-clausal* resolution [2], while *clausal* resolution methods have been investigated by [8, 32].

So, what are the problems in transferring the form of resolution rule outlined above to temporal logics directly? In some cases there are no problems. For example, if the clauses appear within the context of the "always in the future" operator, '$\Box$', then the following version of the resolution rule can be applied.

$$\frac{\Box(A \vee p) \qquad \Box(B \vee \neg p)}{\Box(A \vee B)}$$

In fact, in certain cases, such a rule can even be applied to complementary literals appearing in different temporal contexts. For example, when the "sometime in the future" operator, '$\Diamond$', is used, the following resolution rule can be applied.

$$\frac{\Box(A \vee p) \qquad \Diamond(B \vee \neg p)}{\Diamond(A \vee B)}$$

Thus, several combinations of temporal formulae can be resolved (though formulae such as $\Diamond(A \vee p)$ and $\Diamond(B \vee \neg p)$ can not be resolved in this way as there is no guarantee that the '$A \vee p$' and '$B \vee \neg p$' sub-formulae refer to the same moment in time). As the two operators, '$\Box$' and '$\Diamond$', are the basic connectives of modal logic, clausal resolution rules of the above form have been applied to modal logics [12].

Consequently, the main approaches have either involved the definition of resolution rules for a range of possible combinations of temporal operators, e.g. non-clausal temporal resolution [2], or the use of a normal form such as SNF. The advantage of the latter approach is particularly that, as most of the temporal operators are removed in the translation, fewer resolution rules are required.

Thus, in using SNF, we are not only able to cope with a wide variety of future-time temporal operators, such as $\bigcirc$, $\Box$, and $\Diamond$, but are also able to represent temporal formulae containing *past-time* temporal formulae. Further, SNF can also be extended to some standard modal logics, providing a simple normal form for modal formulae[6].

Given a set of formulae in SNF, we would like to apply resolution rules of the type described above. In particular, it would also be useful to apply the following rule.

$$\frac{\begin{array}{c}\Box(A \vee p) \\ \Diamond(B \vee \neg p)\end{array}}{\Diamond(A \vee B)}$$

Unfortunately, in discrete temporal logics, both the '$\bigcirc$' and '$\Box$' operators can appear, and can interact in an inductive way. In particular, a formula containing both of these operators may contain another hidden $\Box$-formula. For example, consider the formula

$$\Box(a \Rightarrow \bigcirc(l \wedge a)) \ \wedge \ a \ \wedge \ l.$$

It is not immediately obvious that this formula implies $\Box l$, yet temporal resolution applied to the above formula together with $\Diamond\neg l$ should generate a contradiction.

Thus, as well as defining a normal form, a temporal resolution rule that will recognise such $\Box$-formulae must also be provided. Because of the interaction between the '$\bigcirc$' and '$\Box$' operators, recognising $\Box$-formulae requires some form of induction. For example, an inductive argument is required in order to establish that the above formula implies $\Box l$. Such a temporal resolution method, based upon SNF, has been developed [18].

As the formula is now written as a set of SNF rules, the non-temporal part of the resolution process corresponds closely to classical resolution upon these rules. The major *temporal* part of the procedure now consists of recognising hidden $\Box$-formulae within sets of SNF rules. The above formula, conjoined with the contradictory $\Diamond\neg l$ formula is rewritten into the SNF rules shown in Example 1 in §3.5. Since these rules appear in a concise form, it is possible to define a procedure to find such 'hidden' $\Box$-formulae [18]. Indeed, a variety of algorithms have been developed in order to achieve this [10].

4.2 Execution of Temporal Formulae

In recent years a number of programming languages based upon the direct execution of temporal logic formulae have been developed, being utilised particularly in the representation and implementation of a range of dynamic behaviours [21]. A particular family of executable temporal logics uses the normal form developed here as its basis.

The basic idea behind the METATEM framework [4, 17] is to directly use a temporal formula in order to build a model (in our case, a sequence) for that formula. This corresponds to execution in traditional programming languages and provides a language that can be used either for prototyping of temporal specifications, or as a high-level programming language in its own right.

Rather than presenting a detailed description of the METATEM execution mechanism, we will outline how SNF rules can be executed using the basic ideas from METATEM. The intuition behind execution is provided by the simple observation that, in SNF, the initial rules provide constraints upon the initial state, while the *always* and $\Diamond$ rules provide constraints

[6] Once translated to modal logics, the normal form is similar to that provided by Mints [28].

upon the *next* state. Thus, given a program, represented as a set of SNF rules, we can construct the first state in the sequence. We can then construct the next state, given the current state and, by applying this iteratively, can construct the whole state sequence.

Thus, the model structure produced is a sequence of states, with an identified start point, state 0. Execution starts at this initial state, and steps through each state in the structure in turn. When this process is complete (at some trans-finite moment), the structure will be a model for the temporal formula represented by the SNF rules.

To give some idea of the execution process, we will consider the execution of an SNF rule of the form $L \Rightarrow R$ executed at an arbitrary state, t, in the structure that we wish to label. As mentioned in §3.1, we modify SNF slightly by moving all non-temporal literals to the right-hand side of each rule by negating each conjunction. Thus, the left-hand sides of each rule simply represent conditions on the previous state.

We can split the discussion about the execution of our rule into two cases, namely where $t = 0$ and where $t > 0$, as follows.

$t = 0$: As we wish to construct the initial state, we only consider rules of the form

$$\bullet\,\mathbf{false} \;\Rightarrow\; R$$

As R is a disjunction of literals, this provides us with a choice of labellings for the initial state. The execution mechanism simply chooses one of these and continues to construct the next state ($t = 1$).

If the execution mechanism turns out to have made a *bad* choice of disjunctions then execution will eventually backtrack to this choice point.

$t > 0$: As we wish to construct state t, given that $0, \ldots, t-1$ have already been constructed, we consider only non-initial rules. The antecedent of the rule, L, is evaluated in the model at the current state, t. However, since L is guaranteed to be of the form '$\bigcirc M$' (see above), we instead evaluate M in the previous state ($t-1$).

If M evaluates to false, then there is nothing more to be done with this rule, by virtue of the meaning of '$\Rightarrow$', because $\bigcirc\mathbf{false} \Rightarrow R$ is true regardless of R.

If M is evaluated to true, i.e. $\langle \sigma, t-1 \rangle \models M$, then we must ensure that $\langle \sigma, t \rangle \models (\bigcirc M \Rightarrow R)$ is preserved by forcing $\langle \sigma, t \rangle \models R$.

Again, R can either be an eventuality, or a disjunction of literals. If R is an eventuality that can be satisfied, it will be, otherwise it will be recorded. In either case, a choice of labelling is made for t and construction continues on to $t+1$.

When the program consists of several rules, and the interpreter is in a state where the antecedents of more than one rule evaluate to true, the consequents of the successful rules are conjoined to form a single formula which must be made true. Thus if the program consists of rules $L_i \Rightarrow R_i$, and $L_1, \ldots, L_n$ evaluate to true in the current model, then the formula to be made true is $R_1 \wedge \cdots \wedge R_n$.

In this way, a model for the program can be iteratively constructed. Note that, due to the finite model property of PTL, we can constrain the execution so that a model will *eventually* be produced (possibly after a period of backtracking), this is not the case in the first-order extension. In this case, execution is seen just as an *attempt* to build a model [5].

For a longer description of METATEM, including elements such as I/O, strategies for satisfying eventualities, backtracking, loop-checking mechanisms and correctness issues, see [4,

17]. Although much of the development of METATEM has been suspended in favour of Concurrent METATEM [20, 15], the language has applications in system modelling [13], databases [14] and meta-level representation and planning [6].

5 First-Order SNF

In this section, we outline how SNF can be extended to cope with first-order temporal logic. In order to present this, we first extend PTL to a first-order temporal logic, called FTL, which incorporates both the *constant domain* assumption with regard to quantification, and the use of *rigid* designators.

5.1 Syntax of First-Order Temporal Logic

Well-formed formulae of FTL (WFF_f) are generated from the symbols of PTL together with the following.

- A set, $\mathcal{L}_p$, of *predicate symbols* represented by strings of lower-case alphabetic characters.

  Associated with each predicate symbol, p, is a non-negative integer, *arity(p)*.
  (Note that the $\mathcal{L}_p$ associated with PTL represents the subset of the above $\mathcal{L}_p$ consisting of predicates with arity 0.)

- A set, $\mathcal{L}_v$, of *variable symbols*, x, y, z, etc.

- A set, $\mathcal{L}_c$, of *constant symbols*, a, b, c, etc.

- A set, $\mathcal{L}_f$, of *function symbols*, f, g, h, etc.

  Associated with each function symbol, f, is an arity, given by *arity(f)*.

- Quantifiers, $\forall$ and $\exists$.

The set of *terms*, $\mathcal{L}_t$, is defined as follows.

1. Both $\mathcal{L}_v$ and $\mathcal{L}_c$ are subsets of $\mathcal{L}_t$.
2. If $t_1, \ldots, t_n$ are in $\mathcal{L}_t$, and f is a function symbol of arity n, then $f(t_1, \ldots, t_n)$ is in $\mathcal{L}_t$.

The set of well-formed formulae of FTL (WFF_f) is defined as follows.

1. If $t_1, \ldots, t_n$ are in $\mathcal{L}_t$, and p is a predicate symbol of arity n, then $p(t_1, \ldots, t_n)$ is in WFF_f.
2. if A and B are in WFF_f, then the following are in WFF_f

$$\begin{array}{lllll} \neg A & A \vee B & A \wedge B & A \Rightarrow B & (A) \\ \Diamond A & \Box A & A \mathcal{U} B & A \mathcal{W} B & \bigcirc A \\ \blacklozenge A & \blacksquare A & A \mathcal{S} B & A \mathcal{Z} B & \circledcirc A \quad \bullet A \end{array}$$

3. If A is in WFF_f and v is in $\mathcal{L}_v$, then $\exists v.\ A$ and $\forall v.\ A$ are both in WFF_f.

Sub-classifications of WFF_f are the obvious extensions of those defined for WFF_p.

5.2 Semantics of First-Order Temporal Logic

To provide a model structure for FTL, the model structure for PTL is extended to be

$$\sigma \;=\; \langle \mathbf{N}, \pi_p, \pi_c, \pi_f, \mathcal{D} \rangle$$

where

- $\mathbf{N}$ is the Natural Numbers,
- π_p is now a map from $\mathbf{N} \times \mathcal{L}_p$ to $\mathcal{D}^n \to \{\mathsf{T}, \mathsf{F}\}$, where n is the arity of p,
- π_c is a map from $\mathcal{L}_c$ to $\mathcal{D}$,
- π_f is a map from $\mathcal{L}_f$ to $\mathcal{D}^n \to \mathcal{D}$, where n is the arity of f, and,
- $\mathcal{D}$ is the domain over which quantifiers range.

Thus, for a particular state-index, i, and a particular predicate p of arity n, $\pi_p(i,p)$ represents a map from n-tuples of elements of $\mathcal{D}$ to T or F. Note that the *constant domain* assumption is used, i.e., that $\mathcal{D}$ is constant throughout the model.

Next, *variable assignments* and *term assignments* are defined. A variable assignment is a mapping from $\mathcal{L}_v$ to elements of $\mathcal{D}$. Given a variable assignment, V, and the valuation functions, π_c and π_f, associated with a particular model structure, a term assignment $\tau_{v\pi}$ is a mapping from $\mathcal{L}_t$ to $\mathcal{D}$, defined inductively as follows.

- **if** $c \in \mathcal{L}_c$ **then** $\tau_{v\pi}(c) = \pi_c(c)$
- **if** $f \in \mathcal{L}_f$ **then** $\tau_{v\pi}(f(t_1,\ldots,t_n)) = \pi_f(f)(\tau_{v\pi}(t_1),\ldots,\tau_{v\pi}(t_n)) \qquad [\textit{where arity}(f) = n]$
- **if** $v \in \mathcal{L}_v$ **then** $\tau_{v\pi}(v) = V(v)$

The semantics of a well-formed formula is given with respect to a model structure, a state at which the temporal formula is to be interpreted, and a variable assignment. The satisfaction relation, '$\models$', again relates such tuples to well-formed formulae.

The semantics of a predicate is given by the truth value of the predicate application as defined in the model:

$$\langle \sigma, i, V \rangle \models\; p(x_1,\ldots,x_n) \text{ iff } \; \pi_p(i,p)(\tau_{v\pi}(x_1),\ldots,\tau_{v\pi}(x_n)) \;=\; \mathsf{T}.$$

Finally, the semantics of quantifiers is defined as follows.

$$\langle \sigma, i, V \rangle \models\; \forall x.\; \varphi \qquad \text{iff} \qquad \text{for all } d \in \mathcal{D},\; \langle \sigma, i, V \dagger [x \mapsto d] \rangle \models \varphi$$

$$\langle \sigma, i, V \rangle \models\; \exists x.\; \varphi \qquad \text{iff} \qquad \text{there exists } d \in \mathcal{D},\; \text{such that } \langle \sigma, i, V \dagger [x \mapsto d] \rangle \models \varphi$$

As an interpretation now consists of a triple, comprising model, state-index, and assignment components, a well-formed formula, A, is now said to be satisfied in a particular model, σ, at an index, i, and under a particular variable assignment, V, if, and only if, $\langle \sigma, i, V \rangle \models A$. The definition of satisfiability, given earlier, is similarly extended.

In this paper, closed formulae, i.e. formulae containing no free variables, will mainly be used. In this case, the empty mapping, $[\,]$, is used as the initial variable assignment.

5.3 First-Order SNF

We can now extend the definition of SNF to cover FTL, giving us First-Order SNF [19]. A formula in First-Order SNF (SNF_f) is of the form

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

where '$\overline{x_i}$' represents a vector of variables, $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$. Consequently, $\forall \overline{x_i}$ represents

$$\forall x_{i_1}.\ \forall x_{i_2}.\ \ldots\ \forall x_{i_m}$$

and $P_i(\overline{x_i}, \overline{y_i})$ represents

$$P_i(x_{i_1}, x_{i_2}, \ldots, x_{i_m}, y_{i_1}, y_{i_2}, \ldots, y_{i_k}).$$

Now, in SNF_f, each rule is further restricted to be one of the following.

$$\bullet\,\mathbf{false} \quad \Rightarrow \quad \exists \overline{z}.\ \bigvee_{c=1}^{n} r_c(\overline{x_i}, \overline{z})$$

$$\left[\forall \overline{y}.\ \bigwedge_{a=1}^{l} p_a(\overline{x_i}, \overline{y}) \wedge \bigcirc \bigwedge_{b=1}^{m} q_b(\overline{x_i}, \overline{y})\right] \quad \Rightarrow \quad \exists \overline{z}.\ \bigvee_{c=1}^{n} r_c(\overline{x_i}, \overline{z})$$

$$\left[\forall \overline{y}.\ \bigwedge_{a=1}^{l} p_a(\overline{x_i}, \overline{y}) \wedge \bigcirc \bigwedge_{b=1}^{m} q_b(\overline{x_i}, \overline{y})\right] \quad \Rightarrow \quad \exists \overline{z}.\ \Diamond s(\overline{x_i}, \overline{z})$$

where each p_a, q_b, r_c or s is a literal. (Note the similarity between the original SNF and SNF_f.)

Before describing the extra transformation rules for transforming FTL to SNF_f (over those given in §3.1), we will outline some of the problems traditionally encountered in producing a normal-form for first-order temporal (and, indeed, modal) logics.

Some problems with quantifiers

The major problem encountered in first-order temporal (and modal) logics is the interaction between quantifiers and temporal operators. In particular, equivalences such as

$$\Diamond \forall x.\ p(x) \quad \Leftrightarrow \quad \forall x.\ \Diamond p(x)$$

do not, in general, hold. However, *renaming* can again be used to replace certain subformulae. For example, the formula $\Diamond \forall x.\ p(x)$ can be rewritten as

$$\Diamond y \ \wedge\ \Box(y \Leftrightarrow \forall x.\ p(x))$$

Note that the new subformula defining the value of the proposition 'y' is itself in an 'SNF-like' form. We use this property to define renaming transformations for FTL in the next section.

5.4 First-Order Renaming Transformations

We can state a first-order version of the *Renaming Theorem* (Theorem 6), as follows.

Theorem 9 (First-Order Renaming) Given a formula, $\mathcal{A}(B(\overline{x}))$, where $\mathcal{A}$ is a context and B is a formula in WFF_f with free variables, $\overline{x}$, then let C be the formula

$$\mathcal{A}(b(\overline{x})) \wedge \Box\forall\overline{y}.\ (b(\overline{y}) \Leftrightarrow B(\overline{y}))$$

where 'b' is a new predicate symbol and $|\overline{x}| \leq |\overline{y}|$. Given these constraints,

1. if $\mathcal{A}(B(\overline{x}))$ is satisfiable, then C is satisfiable, i.e.
$$\text{if} \quad \exists\sigma.\ \exists V.\ \langle\sigma,0,V\rangle \models \mathcal{A}(B(\overline{x})) \qquad \text{then} \quad \exists\sigma'.\ \exists V'.\ \langle\sigma',0,V'\rangle \models C,$$
and,
2. any model for C is a model for $\mathcal{A}(B(\overline{x}))$, i.e.
$$\forall\sigma.\ \forall V.\ \text{if} \quad \langle\sigma,0,V\rangle \models C \qquad \text{then} \quad \langle\sigma,0,V\rangle \models \mathcal{A}(B(\overline{x}))$$

The transformation process, together with its correctness arguments, follow those presented for SNF in §3.2 and §3.3. Rather than detail the whole process, we just mention a few of the transformation rules based upon renaming.

The following two transformations can be used to remove quantifiers from embedded temporal contexts. (Note that, in the transformation rules that follow, the variables $\overline{x}$ and $\overline{y}$ are assumed to be universally quantified across the formulae.)

$$\{\ P(\overline{x}) \Rightarrow \mathcal{F}(\forall w.\ B(\overline{x},w))\ \} \quad\longrightarrow\quad \left\{ \begin{array}{lcl} P(\overline{x}) & \Rightarrow & \mathcal{F}(b(\overline{x})) \\ b(\overline{y}) & \Leftrightarrow & \forall w.\ B(\overline{y},w) \end{array} \right\}$$

$$\{\ Q(\overline{x}) \Rightarrow \mathcal{F}(\exists z.\ C(\overline{x},z))\ \} \quad\longrightarrow\quad \left\{ \begin{array}{lcl} Q(\overline{x}) & \Rightarrow & \mathcal{F}(c(\overline{x})) \\ c(\overline{y}) & \Leftrightarrow & \exists z.\ C(\overline{y},z) \end{array} \right\}$$

Similarly, we can provide transformations which can be used to ensure that there are no embedded quantifiers on the left-hand side of a rule. These rules are the duals of those given above. For example, embedded existential quantifiers can be removed using the following rule (where $\mathcal{P}$ is a general past-time context).

$$\{\ \mathcal{P}(\exists w.\ B(\overline{x},w)) \Rightarrow F(\overline{x})\ \} \longrightarrow \left\{ \begin{array}{lcl} \mathcal{P}(b(\overline{x})) & \Rightarrow & F(\overline{x}) \\ \exists w.\ B(\overline{y},w) & \Leftrightarrow & b(\overline{y}) \end{array} \right\}$$

The other transformation rules, for example those relating to the removal of temporal operators, carry over from their propositional versions.

5.5 Examples

In order to give a flavour of the FTL to SNF_f translation, we provide the following two examples.

Example 1

Given the formula $\Box\forall x.\ p(x) \Rightarrow \Diamond(\forall z.\ \Box q(x,z))$, the above transformations can be applied to give

$$\begin{array}{llll} \Box\forall x. & p(x) & \Rightarrow & \Diamond b(x) \\ & & \wedge & \\ \Box\forall x. & b(x) & \Leftrightarrow & \forall z.\ \Box q(x,z) \end{array}$$

Then the temporal formula, $\Box q(x,z)$, can be removed from the context of the quantifier, and the resulting formulae can be put in rule form, giving

$$\begin{array}{lll} p(x) & \Rightarrow & \Diamond b(x) \\ b(x) & \Rightarrow & c(x,z) \\ \neg b(x) & \Rightarrow & \exists z.\ \neg c(x,z) \\ c(y,w) & \Rightarrow & q(y,w) \\ c(y,w) & \Rightarrow & a(y,w) \\ \bigcirc a(t,v) & \Rightarrow & q(t,v) \\ \bigcirc a(t,v) & \Rightarrow & a(t,v) \\ \neg c(y,w) & \Rightarrow & \Diamond\neg q(y,w) \end{array}$$

Example 2

In this example, we will show how, as in the propositional case, '$\Rightarrow$' can be used in renaming, rather than '$\Leftrightarrow$'. We will start with a formula that is already in the 'past implies future' form:

$$\Box\forall x.\ [\bigcirc\exists z.\ c(z,x)] \Rightarrow [a(x)\,\mathcal{W}\,(\forall y.\ b(x,y))]$$

First, the embedded quantifier is removed from the future-time component, generating the following rules (again, we assume that all variables not explicitly bound are universally bound at the outer level).

$$\begin{array}{lrcl} 1. & [\bigcirc\exists z.\ c(z,x)] & \Rightarrow & a(x)\,\mathcal{W}\,q(x) \\ 2. & q(x) & \Rightarrow & \forall y.\ b(x,y) \end{array}$$

Similarly, the embedded quantifier is removed from the past-time component, giving the following.

$$\begin{array}{lrcl} 1. & \bigcirc e(x) & \Rightarrow & a(x)\,\mathcal{W}\,q(x) \\ 3. & \neg e(x) & \Rightarrow & \forall z.\ \neg c(z,x) \end{array}$$

All the quantifiers now occur at the outer level of each component, so the '$\mathcal{W}$' operator can be replaced by its fixpoint definition, giving the following.

$$\begin{array}{lrcl} 1. & \bigcirc e(x) & \Rightarrow & q(x) \vee (a(x) \wedge u(x)) \\ 4. & \bigcirc u(x) & \Rightarrow & q(x) \vee (a(x) \wedge u(x)) \end{array}$$

Next, the universal quantifiers that appear in the future-time components of rules 2 and 3 are moved to the outer level, giving the following replacements for these rules. (Note that the variables y and z are now implicitly universally quantified at the outer level.)

$$\begin{array}{lrcl} 2. & q(x) & \Rightarrow & b(x,y) \\ 3. & c(z,x) & \Rightarrow & e(x) \end{array}$$

Finally, rules 1 and 4 are split to ensure that the future-time component of each rule is a disjunction of literals. This splits rule 1 into new rules 1a and 1b, and splits rule 4 into the new rules 4a and 4b. Thus, the final set of SNF_f rules produced from the original formula is

$$
\begin{array}{lrcl}
1a. & \bigcirc e(x) & \Rightarrow & q(x) \vee a(x) \\
1b. & \bigcirc e(w) & \Rightarrow & q(w) \vee u(w) \\
2. & q(x) & \Rightarrow & b(x,y) \\
3. & c(z,x) & \Rightarrow & e(x) \\
4a. & \bigcirc u(y) & \Rightarrow & q(y) \vee a(y) \\
4b. & \bigcirc u(z) & \Rightarrow & q(z) \vee u(z)
\end{array}
$$

Recall that the scope of each variable is restricted to a single rule. Thus, the variable 'x' in rule 2 is not the same as the variable x in rule 3.

5.6 Properties of SNF_f

Any arbitrary FTL formulae can be transformed into SNF_f. This preserves analogous properties to those of the transformation to SNF, namely that the procedure

1. preserves satisfiability, and,
2. any model for the transformed formula is a model for the original formula.

6 Conclusions

In this paper a normal form for discrete, linear temporal logic has been presented. The procedure by which arbitrary formulae are transformed into this normal form has been investigated. This procedure has been implemented as part of a number of systems, involving both temporal proof and temporal execution.

The key observation about this normal form is that a range of temporal attributes can be represented using only a simple set of operators. These *core* operators allow us to describe properties of the current state, of the transitions that can occur between the current and the *next* state, and of situations that will occur at some, unspecified, state in the future. We argue that these aspects represent the essential features of dynamic systems. Further, we have shown how, by representing temporal formulae in this way, both proof and execution mechanisms become both simpler to state (and implement) and more readily understood.

6.1 Related Work

Manna and Pnueli have shown that every PTL formula can be written as a conjunction of a safety and a liveness formula [27]. SNF extends this further so show that the safety formula can be given as a set of transition rules (i.e. $\Box$-formulae), while the liveness formula can be characterised by non-nested '$\Diamond$' operators.

Cavali and del Cerro [8] provided an alternative normal form for temporal formulae and applied it to temporal resolution. However, their normal form is not only more complex than the one described here, but also not as easy to extend to full temporal logic. Similarly, Venkatesh [32] provides a normal form for use with a clausal proof method. Again, our normal form is both simpler and extends to past-time temporal formulae.

Related normal forms have been developed for modal logics, in particular by Enjalbert and del Cerro [12], and by Mints [28].

6.2 Future Work

There are several obvious areas leading from this work. For example, more work is required on refining the complexity bounds for the transformation process. Related to this is the question of how much these bounds can be reduced by using more specific renaming techniques (such as using '$\Rightarrow$' rather than '$\Leftrightarrow$' in certain cases [29]). Although a translator from FTL to SNF_f has already been produced, the *efficient* implementation of the transformation process would then also be undertaken.

The transformations described in this paper have been applied to a specific temporal logic. A particularly important area of future work involves the development of varieties of SNF for extensions of PTL. For example, we are considering alternative temporal models, such as *dense* temporal logics [7], which require a version of SNF with both '$\mathcal{S}$' and '$\mathcal{U}$' as the basic operators, and temporal logics with *infinite past*, where SNF is extended to incorporate '$\blacklozenge$'. We are also investigating the translation of more expressive propositional temporal languages, such as μtl [3], to SNF.

An important direction to explore is what, if any, are the useful restrictions of SNF (e.g., temporal horn clauses [1]). We are looking at the definition of these restrictions, their expressive power and the complexity of the transformation from arbitrary formulae to these restricted classes.

Finally, the applications developed using SNF continue to be investigated. Work both on more refined temporal resolution techniques, and on improved execution algorithms continues.

7 Acknowledgements

The author wishes to thank Clare Dixon, Rajeev Goré, Rob Johnson, Brian Monahan, Philippe Noël, Richard Owens, Martin Peim, Mike Wooldridge and the two anonymous referees for their helpful comments on this work.

References

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

[2] M. Abadi and Z. Manna. Nonclausal Deduction in First-Order Temporal Logic. *ACM Journal*, 37(2):279–317, April 1990.

[3] B. Banieqbal and H. Barringer. Temporal Logic with Fixed Points. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Colloquium on Temporal Logic in Specification*, pages 62–74, Altrincham, U.K., 1987. (Published in *Lecture Notes in Computer Science*, volume 398, Springer Verlag).

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

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

[6] H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. 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.

[7] J. Burgess. Axioms for Tense Logic; 1—"Since' and 'Until". Notre Dame Journal of Formal Logic, 23(4):367–374, October 1982.

[8] A. Cavali and L. Fariñas del Cerro. A Decision Method for Linear Temporal Logic. In R. E. Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, pages 113–127, 1984. (Published in *Lecture Notes in Computer Science*, volume 170, Springer Verlag).

[9] T. Boy de-la Tour and G. Chaminade. The Use of Renaming to Improve the Efficiency of Clausal Theorem Proving. In Ph. Jorrand and V. Sgurev, editors, *Artificial Intelligence IV: Methodology, Systems, Applications*, pages 3–12. Elsevier Science Publishers B.V. (North-Holland), 1990.

[10] C. Dixon. Search Strategies for Resolution in Temporal Logics. In *Thirteenth International Conference on Automated Deduction (CADE)*, New Jersey, USA, July 1996. (Published by Springer Verlag).

[11] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.

[12] P. Enjalbert and L. Fariñas del Cerro. Modal Resolution in Clausal Form. *Theoretical Computer Science*, 65:1–33, 1989.

[13] 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-93)*, Edinburgh, U.K., June 1993. Gordon and Breach Publishers.

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

[15] M. Fisher. 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).

[16] M. Fisher and P. Noël. 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.

[17] 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 *Lecture Notes in Computer Science*, volume 624, Springer-Verlag).

[18] M. Fisher. A Resolution Method for Temporal Logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, Sydney, Australia, August 1991. Morgan Kaufman.

[19] M. Fisher. 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).

[20] M. Fisher. Concurrent METATEM — A Language for Modeling Reactive Systems. In *Parallel Architectures and Languages, Europe (PARLE)*, Munich, Germany, June 1993. (Published in *Lecture Notes in Computer Science*, volume 694, Springer Verlag).

[21] M. Fisher. An Introduction to Executable Temporal Logics. *Knowledge Engineering Review*, 11(1):43–56, March 1996.

[22] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proceedings of the Seventh ACM Symposium on the Principles of Programming Languages*, pages 163–173, Las Vegas, Nevada, January 1980.

[23] D. Gabbay. 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*, pages 402–450, Altrincham, U.K., 1987. (Published in *Lecture Notes in Computer Science*, volume 398, Springer-Verlag).

[24] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, May 1968.

[25] R. Kowalski. Predicate Logic as a Programming Language. In *Proc. IFIP*, pages 569–574. North Holland, 1974.

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

[27] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proc. ACM Symposium on Principles of Distributed Computing*, 1990.

[28] G. Mints. Gentzen-Type Systems and Resolution Rules, Part I: Propositional Logic. *Lecture Notes in Computer Science*, 417:198–231, 1990.

[29] D. A. Plaisted and S. A. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.

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

[31] 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.

[32] G. Venkatesh. A Decision Method for Temporal Logic based on Resolution. In *Lecture Notes in Computer Science*, volume 206, pages 272–289, Springer Verlag, 1986.

[33] P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Stanford University, 1982.

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

[35] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning – Introduction and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.