Is There a Future for Deductive Temporal Verification?

Clare Dixon, Michael Fisher and Boris Konev

Department of Computer Science
The University of Liverpool,
Liverpool L69 3BX, United Kingdom
{C.Dixon,M.Fisher,B.Konev}@csc.liv.ac.uk

Keywords: fragments of PTL; deductive verification; complexity; clausal temporal resolution.

Abstract

In this paper, we consider a tractable sub-class of propositional linear time temporal logic, and provide a complete clausal resolution calculus for it. The fragment is important as it captures simple Büchi automata. We also show that, just as the emptiness check for a Büchi automaton is tractable, the complexity of deciding unsatisfiability, via resolution, of our logic is polynomial (rather than exponential). Consequently, a Büchi automaton can be represented within our logic, and its emptiness can be tractably decided via deductive methods. This may have a significant impact upon approaches to verification, since techniques such as model checking *inherently depend on the ability to check emptiness of an appropriate Büchi automaton. Thus, we also discuss how such a logic might form the basis for* practical *deductive temporal verification.*

1 Introduction

It is widely recognised that *model checking* is the most appropriate verification method for (finite state) systems. Yet there are some surprising aspects to this. The model checking (i.e. satisfiability checking) problem for propositional, linear temporal logic (PTL) is PSPACE-complete [20, 23] yet practical tools for model checking formulae in this logic have been developed, most notably Spin [13]. This has led to deeper investigations into the structure of temporal formulae and their relationship with model checking, most notably the paper by Demri and Schnoebelen [5]. There, the authors consider sub-fragments of PTL, particularly those restricting the number of propositions, the temporal operators allowed, and the depth of temporal nesting in formulae. Demri and Schnoebelen show that, since the formulae tackled in practical model checking often fall within such fragments, then this provides a natural explanation for the viability of model checking in PTL.

Our research has followed a different, but related, direction. Over a number of years, we have been concerned with developing a clausal resolution calculus for both propositional and first-order linear temporal logics [4, 11, 14, 15, 16]. Since deciding unsatisfiability of PTL is also PSPACE-complete, then deductive verification of PTL formulae would seem to be an impractical way to proceed. However, just as Demri and Schnoebelen showed how PTL model checking can be seen as being tractable when we consider fragments of PTL, so we have been examining fragments of PTL that allow clausal resolution to be tractable. In previous work, we examined a fragment where temporal formulae in the clausal form (SNF; see Section 2) were essentially restricted to Horn Clauses [8]. In this paper, however, we investigate a different fragment, where clauses inherently involve XOR operators.

As we will show, the use of XOR has several benefits. Since the complexity of unsatisfiability for XOR clauses in classical propositional logic is low [19], there is the potential to carry much of this over to the temporal case. More importantly, if we consider a Büchi automaton, then we can easily represent the states (using say q_i) and labels (using say l_j) of the automaton in PTL. Indeed, the clausal form we use makes this simpler still with clauses such as[1]

$$(q_1 \wedge l_2) \Rightarrow \bigcirc q_2$$

corresponding directly to transitions (in this case, from state s_1 to state s_2 reading label l_2). However, in such a translation, an underlying problem is representing the fact that the automaton must be in *exactly one* state at any moment in time (and, similarly, that the automaton can only read exactly one label at any moment). This provides an obvious motivation for allowing XOR clauses, since the formula

$$\Box(q_1 \oplus q_2 \oplus \ldots \oplus q_n)$$

captures the property on states that we require.

Thus, in this paper we provide several results. First, we introduce the PTL fragment to be considered and show a completed clausal resolution system for this. Then we show that the complexity of deciding unsatisfiability via resolution is polynomial and, since Büchi automata can be described simply by clauses in this logic, then an emptiness check for a Büchi automaton can be tractably carried out using clausal temporal resolution.

The paper is organised as follows. Section 2 reviews the syntax and semantics of PTL, together with the normal form, SNF, for this logic. In Section 3 we introduce the restriction based on XOR clauses and provide a corresponding modification of SNF. Section 4 introduces the resolution calculus for this restricted logic, and considers the completeness of this calculus, while Section 5 addresses its complexity. In Section 6 we show how Büchi Automata can be translated into this fragment and, in Section 7 we provide concluding remarks, incorporating both related and future work.

2 PTL and SNF

The particular variety of temporal logic we consider is called PTL [12], and is based on a linear, discrete model of time with finite past and infinite future. Although many variations on this simple logic have been examined, we will just use basic PTL with future-time temporal operators.

[1]Here, each proposition, q_i, represents the fact that the automaton is in state s_i. We assume that the automaton has n such states.

2.1 Syntax of PTL

The future-time temporal connectives that we use include $\Diamond$ (*sometime in the future*), $\Box$ (*always in the future*), $\bigcirc$ (*in the next moment in time*), $\mathcal{U}$ (*until*), and $\mathcal{W}$ (*unless*, or *weak until*). Formally, PTL formulae are constructed from the following elements:

- a set, $\mathcal{P}$, of propositional symbols;
- propositional connectives, true, false, $\neg$, $\vee$, $\wedge$, and $\Rightarrow$; and
- temporal connectives, $\bigcirc$, $\Diamond$, $\Box$, $\mathcal{U}$, and $\mathcal{W}$.

The set of well-formed formulae of PTL, denoted by WFF, is inductively defined as the smallest set satisfying the following.

- Any element of $\mathcal{P}$ is in WFF.
- true and false are in WFF.
- If A and B are in WFF then so are

$$\neg A \quad A \vee B \quad A \wedge B \quad A \Rightarrow B$$
$$\Diamond A \quad \Box A \quad A \mathcal{U} B \quad A \mathcal{W} B \quad \bigcirc A\,.$$

A *literal* is defined as either a proposition symbol or the negation of a proposition symbol. An *eventuality* is defined as a well-formed formula of the form $\Diamond A$.

2.2 Semantics of PTL

As discussed above, a sequence of distinct "moments" in time underlie PTL. Typically, the Natural Numbers, $\mathbb{N}$, is used to represent these moments in time. So, a model for PTL, σ, can be characterised as a sequence of *states*

$$\sigma = t_0, t_1, t_2, t_3, \ldots$$

where each state, t_i, is a set of proposition symbols, representing those proposition symbols which are satisfied in the i^{th} moment in time. As formulae in PTL are interpreted at a particular state in the sequence (i.e., at a particular moment in time), the notation

$$(\sigma, i) \models A$$

denotes the truth (or otherwise) of formula A in the model σ at state index $i \in \mathbb{N}$. For any formula A,

$$
\begin{array}{llll}
(\sigma,i) \models p & \text{iff} & p \in t_i & [\text{where } p \in \mathcal{P}] \\
(\sigma,i) \models \mathsf{true} & & & \\
(\sigma,i) \not\models \mathsf{false} & & & \\
(\sigma,i) \models A \land B & \text{iff} & (\sigma,i) \models A \text{ and } (\sigma,i) \models B & \\
(\sigma,i) \models A \lor B & \text{iff} & (\sigma,i) \models A \text{ or } (\sigma,i) \models B & \\
(\sigma,i) \models A \Rightarrow B & \text{iff} & (\sigma,i) \models \neg A \text{ or } (\sigma,i) \models B & \\
(\sigma,i) \models \neg A & \text{iff} & (\sigma,i) \not\models A & \\
(\sigma,i) \models \bigcirc A & \text{iff} & (\sigma,i+1) \models A & \\
(\sigma,i) \models \Diamond A & \text{iff} & \text{there exists a } k \in \mathbb{N} \text{ such that } k \geqslant i \text{ and } (\sigma,k) \models A & \\
(\sigma,i) \models \Box A & \text{iff} & \text{for all } j \in \mathbb{N}, \text{ if } j \geqslant i \text{ then } (\sigma,j) \models A & \\
(\sigma,i) \models A\mathcal{U}B & \text{iff} & \text{there exists a } k \in \mathbb{N}, \text{ such that } k \geqslant i \text{ and } (\sigma,k) \models B & \\
 & & \qquad \text{and for all } j \in \mathbb{N}, \text{ if } i \leqslant j < k \text{ then } (\sigma,j) \models A & \\
(\sigma,i) \models A\mathcal{W}B & \text{iff} & (\sigma,i) \models A\mathcal{U}B \text{ or } (\sigma,i) \models \Box A &
\end{array}
$$

Figure 1. Semantics of PTL

model σ, and state index $i \in \mathbb{N}$, then either $(\sigma, i) \models A$ holds or $(\sigma, i) \models A$ does not hold, denoted by $(\sigma, i) \not\models A$. The pair (σ, i) can be considered as an interpretation (or valuation) for each formula in WFF. (N.B., we will reason about such interpretations in the completeness proof given later.) If there is some σ such that $(\sigma, 0) \models A$, then A is said to be *satisfiable*. If $(\sigma, 0) \models A$ for all models, σ, then A is said to be *valid* and is written $\models A$. Note that formulae here are interpreted at t_0; this is an *anchored* definition of satisfiability and validity [9].

The semantics of WFF can now be given, as in Figure 1.

2.3 SNF, a Normal Form for PTL

The resolution method that we will use later is clausal, and so works on formulae transformed into a normal form. The normal form, called Separated Normal Form (SNF), comprises formulae that are implications with present-time formulae on the left-hand side and (present or) future-time formulae on the right-hand side. The transformation into the normal form reduces most of the temporal operators to a core set and rewrites formulae to be in a particular form. The transformation into SNF depends on three main operations: the renaming of complex subformulae; the removal of temporal operators; and classical style rewrite operations [10, 11].

To assist in the definition of the normal form we introduce a further (nullary) connective 'start' that holds only at the beginning of time, i.e.,

$$(\sigma, i) \models \mathsf{start} \quad \text{iff} \quad i = 0.$$

This allows the general form of the (clauses of the) normal form to be implications. Now, formulae in SNF are of the general form

$$\Box \bigwedge_i A_i$$

where each A_i is known as a *temporal clause* (analogous to a "clause" in classical logic) and must be one of the following forms with each particular k_a, k_b, l_c, l_d, and l representing a literal.

$$
\begin{array}{lcll}
\mathsf{start} & \Rightarrow & \bigvee_c l_c & (\textit{initial}\text{ clause}) \\
\mathsf{true} & \Rightarrow & \bigvee_c l_c & (\textit{universal}\text{ clause}) \\
\bigwedge_a k_a & \Rightarrow & \bigcirc \bigvee_d l_d & (\textit{step}\text{ clause}) \\
\bigwedge_b k_b & \Rightarrow & \Diamond l & (\textit{sometime}\text{ clause})
\end{array}
$$

For convenience, the outer $\Box$ and $\land$ connectives are usually omitted, and the set of clauses $\{A_i\}$ is considered.

While the translation from arbitrary temporal formulae to SNF will not be described further here, we note that such such a transformation not only preserves satisfiability, but also ensures any model generated from the formula in SNF is a model for the original formula [10]. In addition, the complexity of the translation process is low [11].

3 PTL-X$_{\mathcal{A}}$ and SNFX$_{\mathcal{A}}$

We will now define additional syntax for PTL, namely the XOR operator, '$\oplus$', and characterise a modification of SNF, called *SNFX$_{\mathcal{A}}$*, especially modified to capture *automata-properties*. The key aspect here is that the set of propositions, $\mathcal{P}$, is partitioned into two disjoint sets, $\mathcal{S}$ and $\mathcal{L}$. Note that these will later represent *states* and *labels* once we begin translating automata into SNFX$_{\mathcal{A}}$.

The XOR operator is defined simply as

$(\sigma, i) \models \varphi_1 \oplus \varphi_2 \oplus \ldots \oplus \varphi_m$ iff
there is exactly one $1 \leq j \leq m$ such that $(\sigma, i) \models \varphi_j$.

The new logic, PTL-X$_{\mathcal{A}}$, will comprise exactly those clauses that can be represented in SNFX$_{\mathcal{A}}$. Thus, we will concentrate first on SNFX$_{\mathcal{A}}$. Like SNF, SNFX$_{\mathcal{A}}$ is of the general form

$$\Box \bigwedge_i A_i$$

where each A_i must be one of the following.

$$\begin{array}{lll l}
\mathsf{start} & \Rightarrow & \bigvee_k q_k & (\textit{initial} \text{ clause}) \\
(q_i \wedge l_j) & \Rightarrow & \bigcirc \bigvee_k q_k & (\textit{step} \text{ clause}) \\
\mathsf{true} & \Rightarrow & R_c & (\textit{universal} \text{ clause}) \\
\mathsf{true} & \Rightarrow & \Diamond \bigvee_k q_k & (\textit{sometime} \text{ clause}) \\
\mathsf{true} & \Rightarrow & q_1 \oplus q_2 \oplus \ldots \oplus q_n & (\textit{XOR-}\mathcal{S} \text{ clause}) \\
\mathsf{true} & \Rightarrow & l_1 \oplus l_2 \oplus \ldots \oplus l_m & (\textit{XOR-}\mathcal{L} \text{ clause})
\end{array}$$

where $q_i, q_k \in \mathcal{S}$ and $l_j \in \mathcal{L}$, and where R_c must be one of $\neg q_i$, or $(\neg q_i \vee \neg l_j)$.

In SNFX$_{\mathcal{A}}$, at most one *sometime* clause and at most one *initial* clause is allowed. $\mathcal{S}$ must equal $\{q_1, q_2, \ldots, q_n\}$ and $\mathcal{L}$ must equal $\{l_1, l_2, \ldots, l_m\}$. Thus, all elements of $\mathcal{S}$ and $\mathcal{L}$ occur within some XOR clause. In addition, there is a further restriction on the form above, namely that, for every q_i, l_j such that $q_i \in \mathcal{S}$ and $l_j \in \mathcal{L}$ there is at most one clause of the form

$$(q_i \wedge l_j) \Rightarrow \bigcirc \bigvee_k q_k$$

in the clause set.

4 Clausal Temporal Resolution for SNFX$_{\mathcal{A}}$

Next we consider resolution rules for sets of SNFX$_{\mathcal{A}}$ clauses. The resolution rules are split into four groups: initial resolution; step resolution; hyper XOR resolution and temporal resolution.

Initial Unit Resolution involves resolving an initial clause with a universal clause:

$$\boxed{\mathsf{IURES}} \quad \begin{array}{lcl}
\mathsf{start} & \Rightarrow & Q \vee q_i \\
\mathsf{true} & \Rightarrow & \neg q_i \\
\hline
\mathsf{start} & \Rightarrow & Q
\end{array}$$

The conclusion of the rule, $\mathsf{start} \Rightarrow Q$ replaces the premise $\mathsf{start} \Rightarrow Q \vee q_i$.

Step Resolution resolves step clauses with universal clauses (Step Unit Resolution, SURES) or derives additional universal clauses from contradictions obtained in the next moment (SRES):

$$\boxed{\mathsf{SURES}} \quad \begin{array}{lcl}
q_i \wedge l_j & \Rightarrow & \bigcirc(Q \vee q_k) \\
\mathsf{true} & \Rightarrow & \neg q_k \\
\hline
q_i \wedge l_j & \Rightarrow & \bigcirc Q
\end{array}$$

The conclusion of the rule, $q_i \wedge l_j \Rightarrow \bigcirc Q$ replaces the premise $q_i \wedge l_j \Rightarrow \bigcirc(Q \vee q_k)$.

$$\boxed{\mathsf{SRES}} \quad \begin{array}{lcl}
q_i \wedge l_j & \Rightarrow & \bigcirc \mathsf{false} \\
\hline
\mathsf{true} & \Rightarrow & \neg q_i \vee \neg l_j
\end{array}$$

Hyper XOR Resolution takes several universal clauses relating to the negation of a proposition in $\mathcal{S}$, together with the XOR-$\mathcal{L}$ clause:

$$\boxed{\mathsf{HRES}} \quad \begin{array}{lcl}
\mathsf{true} & \Rightarrow & \neg q_k \vee \neg l_1 \\
\ldots & \Rightarrow & \ldots \\
\mathsf{true} & \Rightarrow & \neg q_k \vee \neg l_m \\
\mathsf{true} & \Rightarrow & l_1 \oplus \ldots \oplus l_m \\
\hline
\mathsf{true} & \Rightarrow & \neg q_k
\end{array}$$

The conclusion of the rule, $\mathsf{true} \Rightarrow \neg q_k$ replaces the first m premises (of the form $\mathsf{true} \Rightarrow \neg q_k \vee \neg l_j$).

Temporal Resolution Since there is only one sometime clause which is of a simple form (i.e. it has true on the left hand side) we can use a simplified version of the standard [11] step resolution rule, defined in [3]:

$$\boxed{\mathsf{TRES}} \quad \begin{array}{rcl} \bigvee_j q_j & \Rightarrow & \Box \bigwedge_k \neg q_k \\ \mathsf{true} & \Rightarrow & \Diamond \bigvee_k q_k \\ \hline \mathsf{true} & \Rightarrow & \bigwedge_j \neg q_j \end{array}$$

To apply TRES we must find a (non-temporal) formula $\bigvee_j q_j$ such that $\bigvee_j q_j$ implies $\Box \bigwedge_k \neg q_k$.

For standard SNF clauses this problem has been addressed previously in [6]. Here we have a simpler set of clauses so the search for a *loop* (i.e. a set of clauses that imply $\Box \bigwedge_k \neg q_k$) is easier.

Loop Search Assume we are resolving with $\mathsf{true} \Rightarrow \Diamond \bigvee_k q_k$. Let $E = \{q_k \mid q_k \in \bigvee_k q_k\}$.

- Construct a set SC which initially contains the set of step clauses.
- Create two sets of propositions: L_G, representing *good* propositions, and L_B, representing *bad* propositions. Initially, let L_G be the members of $\mathcal{S}$ which occur on the left hand sides of clauses in SC which are not in E and let $L_B = \mathcal{S} \setminus L_G$.
- Iteratively search through SC for clauses $q_k \wedge l_a \Rightarrow \bigcirc(Q \vee q_b)$ where $q_b \in L_B$ or clauses $q_k \wedge l_a \Rightarrow \bigcirc\mathsf{false}$. Delete $q_k \wedge l_a \Rightarrow \bigcirc(Q \vee q_b)$ (respectively $q_k \wedge l_a \Rightarrow \bigcirc\mathsf{false}$) from SC, delete q_k from L_G and and let $L_B = L_B \cup \{q_k\}$.
- Terminate when either $SC = \emptyset$ or SC doesn't change as we search through the clauses.
- If $SC = \emptyset$ there is no loop, otherwise the loop is $\bigvee_{q \in L_G} q \Rightarrow \Box \bigwedge_k \neg q_k$.

Subsumption Finally, we assume that standard subsumption takes place.

Since the SNFX$_{\mathcal{A}}$ temporal resolution rules can be seen as a particular strategy for unrestricted temporal resolution [11] (note that in both unit resolution rules, the conclusion of the rule subsumes the premise); we, obviously, have the following soundness theorem.

Theorem 1 *The rules of clausal temporal resolution preserve satisfiability.*

The completeness theorem requires a proof.

Theorem 2 *If a set of SNFX$_{\mathcal{A}}$ clauses is unsatisfiable then the temporal resolution procedure will derive a contradiction when applied to it.*

Proof
We adapt the completeness proof of the original system [11, 3] as described below. First, we introduce additional definitions.

We split the set of temporal clauses into four groups. Let

$\mathcal{I}$ denote the *initial* clause,

$\mathcal{U}$ be the set of all *universal* clauses,

$\mathcal{T}$ be the set of all *step* clauses,

$\mathcal{E}$ be the *sometime* clause, and

$\mathcal{X}$ be the set of *XOR* clauses.

Definition 3 (behaviour graph) *Given a set of SNFX$_{\mathcal{A}}$ clauses over a set of propositional symbols $\mathcal{P}$, we construct a finite directed graph G as follows. The nodes of G are interpretations of $\mathcal{P}$, and an interpretation, I, representing some pair (σ, i), is a node of G if $I \models \mathcal{U} \cup \mathcal{X}$.*

For each node, I, we construct an edge in G to a node I' if, and only if, the following condition is satisfied:

- *For every step clause $(P \Rightarrow \bigcirc Q) \in \mathcal{T}$, if $I \models P$ then $I' \models Q$.*

A node, I, is designated an initial node of G if $I \models \mathcal{I} \cup \mathcal{U} \cup \mathcal{X}$. The behaviour graph *$H$ of the set of clauses is the maximal subgraph of G given by the set of all nodes reachable from initial nodes.*

Notice that, because of the XOR-clauses, exactly one proposition $q \in \mathcal{S}$ and exactly one proposition $l \in \mathcal{L}$ are true in I. Therefore, we can associate nodes of the behaviour graph, H, with pairs (q, l), where $q \in \mathcal{S}$ and $l \in \mathcal{L}$.

Let (q, l), (q', l') be nodes of graph H. We denote the relation

"(q', l') is an immediate successor of (q, l)"

by $(q, l) \to (q', l')$, and the relation

"(q', l') is a successor of (q, l)"

by $(q, l) \to^{+} (q', l')$.

The proof of completeness proceeds by induction on the number of nodes in the behaviour graph H, which is finite. If H is empty then the set $\mathcal{U} \cup \mathcal{I} \cup \mathcal{X}$ is unsatisfiable. In this case there must exist a derivation by IURES and HRES (and this is because the rules IURES and HRES taken alone coincide with complete classical hyper resolution).

Now suppose H is not empty. Let I be a node of H which has no successors. We show that there exists an inference by temporal resolution deleting the node from the graph. Then, there exists exactly one step rule

$$q \wedge l \Rightarrow \bigcirc \bigvee_{k} q_k,$$

whose left-hand side matches (q, l). Notice that, for every k and every $j \in \{1, \dots m\}$, we have $(q_k \wedge l_j) \wedge \mathcal{U} \vdash \bot$ (for otherwise, there would be an edge in H from (q, l) to (q_k, l_j)). Because of the restricted form of U, it means that for every $j \in \{1, \dots m\}$, we have $\neg q_k \vee \neg l_j \in \mathcal{U}$. Therefore, for every k the clause $\mathsf{true} \Rightarrow \neg q_k$ can be deduced by HRES and, hence, the clause $\mathsf{true} \Rightarrow \neg q \vee \neg l$ can be obtained by SURES, SRES. This eliminates node I from the behaviour graph.

In case when all nodes of H have a successor, a contradiction can be derived with the help of the temporal resolution rule TRES. Note that we impose no restriction on this rule (it coincides with the temporal resolution rule for the general calculi presented in [11, 3]) and the proof of completeness is no different from what is already published [11, 3]. □

5 Complexity of SNFX$_{\mathcal{A}}$ Resolution

To analyse the complexity of SNFX$_{\mathcal{A}}$ resolution, we first consider the complexity of the saturation procedure by step resolution (by step resolution we mean rules IURES, SURES, SRES, and HRES), then we consider the complexity of loop search, and finally, we consider the overall complexity of the proof procedure.

- Complexity of step resolution

  Let $\mathcal{C}$ be a set of SNFX$_{\mathcal{A}}$ clauses. Recall that the set of propositions in $\mathcal{C}$ is partitioned into two disjoint sets, $\mathcal{S}$ and $\mathcal{L}$; let the cardinality of $\mathcal{S}$ be n the cardinality of $\mathcal{L}$ be m.

  We show that there exists a polynomial-complexity (in terms of n and m) procedure that saturates $\mathcal{C}$ by step resolution, that is, applies the rules IURES, SURES, SRES, and HRES to $\mathcal{C}$ exhaustively until no new clause can be derived.

  Notice that any saturation procedure, which ensures that no inference rule is attempted on the same set of premises more than once, will have a polynomial complexity. Notice further that the Given Clause Algorithm [18] satisfies this requirement.

  The complexity of the procedure is bounded then by the number of different sets of premises to which inference rules can be applied. It suffices to notice that the HRES rule can be applied to at most n different sets of premises; SRES to at most $m \times n$ sets of premises; the SURES rule can be applied to at most $n^2 \times m$ sets of different premises (notice that, since no two step clauses have the same left-hand side, there are at most $n \times m$ different step rules in any clause set); and, similarly, the IURES rule can be applied to at most n sets of different premises. Altogether, the complexity of the saturation procedure is $O(n^2 \times m)$.

- Complexity of loop search

  Notice that since at every iteration of loop search, at least one proposition is deleted from L_G, there are at most n iterations. Using efficient implementation techniques, the search in every iteration can be implemented in time bounded by

$n \times m$. Therefore, the complexity of loop search is $n^2 \times m$.

- Overall complexity

  The overall procedure works as follows: the set of clauses is saturated by step resolution, then loop search is attempted. If loop search succeeds, the set of clauses is extended by the conclusion of the TRES rule and the entire process repeats (we call the process the *main loop*) until either a contradiction is obtained, or nothing new can be derived.

  The overall complexity of the proof procedure is bounded by the product of the number of iterations of the main loop and the joint complexity of saturation and loop search. Note that there may not be more than n iterations of the main loop. Therefore, the overall complexity of proof search is $O(n^3 \times m)$.

6 From Büchi Automata to SNFX$_{\mathcal{A}}$

We will now consider the representation of a Büchi automaton as a set of SNFX$_{\mathcal{A}}$ clauses and, in particular, emptiness checking of the automaton as deriving a refutation in SNFX$_{\mathcal{A}}$. We begin with a standard definition of a Büchi automaton [21, 22].

6.1 Definition of a Büchi automaton

A Büchi automaton, $\mathcal{A}$, is a tuple $\langle \Sigma,\ S,\ F_0,\ \delta,\ F\rangle$, where:

- $\Sigma = \{\pi_0, \ldots \pi_m\}$ is a finite non-empty alphabet;
- $S = \{s_0 \ldots s_n\}$ is a finite set of states;
- $F_0 \subseteq S$, is a set of initial states;
- $\delta = S \times \Sigma \longrightarrow 2^S$ is a non-deterministic transition function; and
- $F \subseteq S$, is a set of accepting states.

A *run* $\tau_{\mathcal{A}} = r_0, r_1, r_2, \ldots$ of a Büchi automaton, $\mathcal{A}$, over the word $w = w_0 w_1 w_2 \ldots$, where $w_j \in \Sigma$, is an infinite sequence of states, $r_i \in S$ where the first state is the initial state, i.e. $r_0 \in F_0$, and for every other state r_{i+1} for $i = 0, 1, \ldots$ we have $r_{i+1} \in \delta(r_i, w_i)$.

A run, $\tau_{\mathcal{A}}$, is *accepting* if there is a state $s \in F$ such that s appears in $\tau_{\mathcal{A}}$ infinitely often.

6.2 From Büchi Automata to SNFX$_{\mathcal{A}}$

We aim to construct a set of SNFX$_{\mathcal{A}}$ clauses T from $\mathcal{A}$ such that T is satisfiable if, and only if, $\mathcal{A}$ has an accepting run.

To represent $\mathcal{A}$ in SNFX$_{\mathcal{A}}$ we use the following propositions:-

- q_i for each $s_i \in S$;
- l_j for each $\pi_j \in \Sigma$.

The set $\mathcal{C}_{\mathcal{A}}$ of SNFX$_{\mathcal{A}}$ clauses representing the automata $\mathcal{A}$ is as follows.

$$\begin{array}{rcll}
\mathsf{start} & \Rightarrow & \bigvee\limits_i q_i & \text{for } s_i \in F_0 \\
(q_i \wedge l_k) & \Rightarrow & \bigcirc \bigvee\limits_j q_j & \text{for } s_j \in \delta(s_i, \pi_k) \\
(q_i \wedge l_k) & \Rightarrow & \bigcirc \mathsf{false} & \text{for } \delta(s_i, \pi_k) = \emptyset \\
\mathsf{true} & \Rightarrow & q_1 \oplus \ldots \oplus q_n & \text{for } S = \{s_1, \ldots s_n\} \\
\mathsf{true} & \Rightarrow & l_1 \oplus \ldots \oplus l_m & \text{for } \Sigma = \{\pi_1, \ldots \pi_m\} \\
\mathsf{true} & \Rightarrow & \Diamond \bigvee\limits_j q_j & \text{for } s_j \in F
\end{array}$$

Proposition 4 *A Büchi automaton $\mathcal{A} = \langle \Sigma,\ S,\ F_0,\ \delta,\ F\rangle$ has an accepting run $\tau_{\mathcal{A}}$ (over infinite word w) if, and only if, the set of SNFX$_{\mathcal{A}}$ clauses, $\mathcal{C}_{\mathcal{A}}$, defined above, is satisfiable.*

Proof

$\Rightarrow$) We first show that, given a Büchi automaton, $\mathcal{A}$, with an accepting run such that $\mathcal{C}_{\mathcal{A}}$ is its translation into SNFX$_{\mathcal{A}}$, as described above, there is a model which satisfies $\mathcal{C}_{\mathcal{A}}$.

Let $\mathcal{A} = \langle \Sigma,\ S,\ F_0,\ \delta,\ F\rangle$, be a given non-empty Büchi automaton and let $\mathcal{A}$ have an accepting run $\tau_{\mathcal{A}} = r_0 r_1 r_2 \ldots r_t r_{t+1} \ldots$, ($r_t \in S$ for $t = 0, 1, 2, \ldots$) over an infinite word $w = w_0 w_1 w_2 \ldots w_t w_{t+1} \ldots$. For some accepting state $s_f \in F$, s_f must appear infinitely often in $\tau_{\mathcal{A}}$. In the run $\tau_{\mathcal{A}}$, at the t^{th} moment of time when the automaton is in the state r_t and reads w_t, it moves to r_{t+1}, i.e. $\delta(r_t, w_t) = r_{t+1}$.

We now construct a model σ and show it satisfies the clause set $\mathcal{C}_{\mathcal{A}}$. We note that as $\square(A \wedge B) \equiv \square A \wedge \square B$ we can assume that the external " $\square$

" operator in Section 3 is applied to each implication in $\mathcal{C}_{\mathcal{A}}$.

Let $\mathcal{P}$ be a set of propositional symbols where $\mathcal{P} = \{l_j \mid \pi_j \in \Sigma\} \cup \{q_i \mid s_i \in S\}$. We construct an infinite sequence of states

$$\sigma = u_0,\ u_1,\ u_2,\ \ldots,\ u_t,\ u_{t+1},\ \ldots$$

as follows. Set the propositions that are true in each state to match those read by $\mathcal{A}$ on the accepting run for the infinite word w, i.e. $l_j \in u_t$ if, and only if, $w_t = \pi_j$. For any $q_j \in \mathcal{P}$ then $q_j \in u_t$ if, and only if, $r_t = s_j$ (set q_j to be true if and only if the state visited in the t^{th} moment in time of the accepting run, $\tau_{\mathcal{A}}$, is s_j).

Next we show σ satisfies the clause set $\mathcal{C}_{\mathcal{A}}$.

The run $\tau_{\mathcal{A}}$ is an accepting run which starts from s_0. Thus s_0 is one of the initial states, i.e. $s_0 \in F_0$, and from how we have constructed σ, q_0 is satisfied in the initial moment 0, i.e. $(\sigma, 0) \models q_0$. Also as $(\sigma, 0) \models \mathsf{start}$ and $(\sigma, t) \not\models \mathsf{start}$ for $t > 0$, from the semantics of start, the initial clause of the clause set $C_{\mathcal{A}}$

$$\mathbf{start} \Rightarrow \bigvee_{s_j \in F_0} q_j$$

is satisfied at every moment in time.

Next we must show that the step clauses of $\mathcal{C}_{\mathcal{A}}$ hold. Consider the implication,

$$(q_i \wedge l_k) \Rightarrow \bigcirc \bigvee_j q_j.$$

For any moment t such that $(\sigma, t) \not\models q_i$ or $(\sigma, t) \not\models l_k$ the above holds trivially.

Next consider some time t such that $(\sigma, t) \models q_i$ and $(\sigma, t) \models l_k$. We must show that $(\sigma, t) \models \bigcirc \bigvee_j q_j$. From the construction of σ there must be some state $s_i = r_t$ which is visited in the t_{th} moment of the accepting run and a transition $s_l \in \delta(s_i, \pi_k)$ such that in the $t+1$st moment in time the accepting run is at state $s_l = r_{t+1}$ having read $\pi_k = w_t$. Thus, from the construction of σ, $(\sigma, t+1) \models q_l$ and from how we have constructed $C_{\mathcal{A}}$ and the semantics of $\vee$,

$$(\sigma, t+1) \models \bigvee_j q_j\,.$$

Hence, from the semantics of "$\bigcirc$",

$$(\sigma, t) \models \bigcirc \bigvee_j q_j$$

and

$$(\sigma, t) \models q_i \wedge l_k \Rightarrow \bigcirc \bigvee_j q_j.$$

Thus, at all moments in time each step clause holds and

$$\square \left[(q_i \wedge l_k) \Rightarrow \bigcirc \bigvee_j q_j \right]$$

is satisfied.

Recall that the run $\tau_{\mathcal{A}}$ in the t^{th} moment of time visits the state $s_i = r_t$. From the construction of σ, $(\sigma, t) \vDash q_i$ and $(\sigma, t) \nvDash q_j$ for every $q_j \neq q_i$. Hence the XOR-$\mathcal{S}$ clause is also satisfied in σ at every moment. Similarly from the infinite word w of the accepting run and how we have constructed σ at each state we have $(\sigma, t) \models l_i$ for some l_i such that $1 \leq i \leq m$ and $(\sigma, t) \not\models l_j$ for all $j \neq i$ such that $1 \leq j \leq m$.

Finally consider the sometime clause. From the construction of the model $(\sigma, t) \vDash q_f$ if, and only if, $r_t = s_f$ and since the automaton $\tau_{\mathcal{A}}$ hits the state s_f infinitely often, the sometime clause is satisfied.

Therefore, all clauses in $C_{\mathcal{A}}$ are satisfiable in σ.

$\Leftarrow$) Assume now that for an automaton $\mathcal{A}$, the corresponding set of SNFX$_{\mathcal{A}}$ clauses, C_A, is satisfiable. We show that $\mathcal{A}$ has an accepting run. Consider the sequence of states $\sigma = u_0, u_1, u_2, \ldots$ such that $(\sigma, 0) \models \mathcal{C}_{\mathcal{A}}$. Because of the XOR clauses, for every $t \geq 0$ there is *exactly one* $q_i \in S$ and *exactly one* $l_k \in \Sigma$ such that $q_i \in u_t$ and $l_k \in u_t$. We show by induction on t that the sequence of states $r_0, r_1, \ldots r_t$ and the word $\pi_0, \pi_1, \ldots \pi_t$ are such that $r_0 \in F_0$ and $\delta(s_t, \pi_t) = s_{t+1}$.

For $t = 0$, since

$$(\sigma, 0) \models (\mathsf{start} \Rightarrow \bigvee_{s_j \in F_0} q_j)$$

the state s_0 is initial. Consider now the step clause

$$(q_i \wedge l_k) \Rightarrow \bigcirc \bigvee_j q_j.$$

(Note that the right-hand side of the step clause cannot be false for otherwise $\mathsf{false} \in u_{t+1}$.) Then $q_j \in u_{t+1}$. Note that $s_j \in \delta(s_i, \pi_k)$.

It remains to notice that, since

$$(\sigma, 0) \models \square(\mathsf{true} \Rightarrow \Diamond \bigvee_i q_j)$$

for $s_j \in F$, the state s_j appears in the sequence $\tau_{\mathcal{A}} = s_0, s_1, \ldots, s_t, \ldots$ infinitely often, that is, the run $\tau_{\mathcal{A}}$ is accepting.

□

Example 1 Consider a Büchi Automaton $\mathcal{A}_1 = \langle \Sigma,\ S,\ F_0,\ \delta,\ F\rangle$, where:

- $\Sigma = \{\pi_0, \pi_1\}$
- $S = \{s_1, s_2, s_3, s_4, s_5\}$
- $F_0 = \{s_1\}$
- $F = \{s_1, s_5\}$

The transitions are given below.

S	$\delta(s,\pi_1)$	$\delta(s,\pi_2)$
s_1	$\{s_2, s_5\}$	$\{s_5\}$
s_2	$\{s_4\}$	$\{s_3\}$
s_3	$\{s_2\}$	$\{s_4\}$
s_4	$\{s_3, s_4\}$	$\{s_2\}$
s_5	$\{s_4\}$	$\{s_2\}$

Hence there is no accepting run.

$$\begin{array}{rrcl}
0. & \mathsf{start} & \Rightarrow & q_1 \\
1. & q_1 \wedge l_1 & \Rightarrow & \bigcirc(q_2 \vee q_5) \\
2. & q_1 \wedge l_2 & \Rightarrow & \bigcirc q_5 \\
3. & q_2 \wedge l_1 & \Rightarrow & \bigcirc q_4 \\
4. & q_2 \wedge l_2 & \Rightarrow & \bigcirc q_3 \\
5. & q_3 \wedge l_1 & \Rightarrow & \bigcirc q_2 \\
6. & q_3 \wedge l_2 & \Rightarrow & \bigcirc q_4 \\
7. & q_4 \wedge l_1 & \Rightarrow & \bigcirc(q_3 \vee q_4) \\
8. & q_4 \wedge l_2 & \Rightarrow & \bigcirc q_2 \\
9. & q_5 \wedge l_1 & \Rightarrow & \bigcirc q_4 \\
10. & q_5 \wedge l_2 & \Rightarrow & \bigcirc q_2 \\
11. & \mathsf{true} & \Rightarrow & q_1 \oplus q_2 \oplus q_3 \oplus q_4 \oplus q_5 \\
12. & \mathsf{true} & \Rightarrow & l_1 \oplus l_2 \\
13. & \mathsf{true} & \Rightarrow & \Diamond(q_1 \vee q_5)
\end{array}$$

Loop Search Initially, $L_G = \{q_2, q_3, q_4\}$ and $L_B = \{q_1, q_5\}$. There is no change to either set so the loop is

$$(q_2 \vee q_3 \vee q_4) \Rightarrow \square(\neg q_1 \wedge \neg q_5)$$

Applying temporal resolution we obtain.

$$\begin{array}{rrcll}
14. & \mathsf{true} & \Rightarrow & \neg q_2 & [13, \mathsf{TRES}] \\
15. & \mathsf{true} & \Rightarrow & \neg q_3 & [13, \mathsf{TRES}] \\
16. & \mathsf{true} & \Rightarrow & \neg q_4 & [13, \mathsf{TRES}]
\end{array}$$

Thus clauses 3–8 are subsumed by one of 14–16.

$$\begin{array}{rrcll}
17. & q_5 \wedge l_1 & \Rightarrow & \bigcirc\mathsf{false} & [9, 16, \mathsf{SURES}] \\
18. & q_5 \wedge l_2 & \Rightarrow & \bigcirc\mathsf{false} & [10, 14, \mathsf{SURES}] \\
19. & \mathsf{true} & \Rightarrow & \neg q_5 \vee \neg l_1 & [17, \mathsf{SRES}] \\
20. & \mathsf{true} & \Rightarrow & \neg q_5 \vee \neg l_2 & [18, \mathsf{SRES}] \\
21. & \mathsf{true} & \Rightarrow & \neg q_5 & [12, 19, 20, \mathsf{HRES}] \\
22. & q_1 \wedge l_1 & \Rightarrow & \bigcirc q_5 & [1, 14, \mathsf{SURES}] \\
23. & q_1 \wedge l_1 & \Rightarrow & \bigcirc\mathsf{false} & [21, 22, \mathsf{SURES}] \\
24. & q_1 \wedge l_2 & \Rightarrow & \bigcirc\mathsf{false} & [2, 21, \mathsf{SURES}] \\
25. & \mathsf{true} & \Rightarrow & \neg q_1 \vee \neg l_1 & [23, \mathsf{SRES}] \\
26. & \mathsf{true} & \Rightarrow & \neg q_1 \vee \neg l_2 & [24, \mathsf{SRES}] \\
27. & \mathsf{true} & \Rightarrow & \neg q_1 & [12, 25, 26, \mathsf{HRES}] \\
28. & \mathsf{start} & \Rightarrow & \mathsf{false} & [0, 27, \mathsf{IURES}]
\end{array}$$

Example 2 Now, consider a Büchi Automaton $\mathcal{A}_2 = \langle \Sigma,\ S,\ F_0,\ \delta,\ F\rangle$, where:

- $\Sigma = \{\pi_1\}$
- $S = \{s_1, s_2, s_3\}$

- $F_0 = \{s_1\}$
- $F = \{s_2\}$

The transitions are given below.

S	$\delta(s, \pi_1)$
s_1	$\{s_1, s_2\}$
s_2	$\{s_3\}$
s_3	$\{s_3\}$

Hence there is no accepting run.

$$\begin{array}{rrcl}
0. & \mathsf{start} & \Rightarrow & q_1 \\
1. & q_1 \wedge l_1 & \Rightarrow & \bigcirc(q_1 \vee q_2) \\
2. & q_2 \wedge l_1 & \Rightarrow & \bigcirc q_3 \\
3. & q_3 \wedge l_1 & \Rightarrow & \bigcirc q_3 \\
4. & \mathsf{true} & \Rightarrow & q_1 \oplus q_2 \oplus q_3 \\
5. & \mathsf{true} & \Rightarrow & l_1 \\
6. & \mathsf{true} & \Rightarrow & \Diamond q_2
\end{array}$$

Note that, since the only symbol in the alphabet is π_1, the XOR-$\mathcal{L}$ clause is simply $\mathsf{true} \Rightarrow l_1$ (clause 5).

Loop Search Initially, $L_G = \{q_1, q_3\}$ and $L_B = \{q_2\}$. From clause 1 we delete q_1 from L_G and add to L_B and obtain $L_G = \{q_3\}$ and $L_B = \{q_1, q_2\}$. There is no change to either set so the loop formula is

$$q_3 \Rightarrow \Box\neg q_2$$

By applying temporal resolution we obtain the following.

$$\begin{array}{rrcll}
7. & \mathsf{true} & \Rightarrow & \neg q_3 & [6, \mathsf{TRES}]
\end{array}$$

This subsumes clause 3.

$$\begin{array}{rrcll}
8. & q_2 \wedge l_1 & \Rightarrow & \bigcirc \mathsf{false} & [2, 7, \mathsf{SURES}] \\
9. & \mathsf{true} & \Rightarrow & \neg q_2 \vee \neg l_1 & [8, \mathsf{SRES}] \\
10. & \mathsf{true} & \Rightarrow & \neg q_2 & [5, 9, \mathsf{HRES}] \\
11. & q_1 \wedge l_1 & \Rightarrow & \bigcirc q_1 & [1, 10, \mathsf{SURES}]
\end{array}$$

Clause 11 now subsumes clause 2. Now, attempting loop search again (note the current set of step clauses is just clause 11) we have $L_G = \{q_1\}$ and $L_B = \{q_2, q_3\}$. We obtain the loop

$$q_1 \Rightarrow \Box\neg q_2$$

By applying temporal resolution we obtain the following.

$$\begin{array}{rrcll}
12. & \mathsf{true} & \Rightarrow & \neg q_1 & [6, \mathsf{TRES}] \\
13. & \mathsf{start} & \Rightarrow & \mathsf{false} & [0, 12, \mathsf{IURES}]
\end{array}$$

7 Conclusions

In this paper we have introduced a novel fragment of PTL, and have provided a complete resolution calculus for this fragment. The complexity analysis carried out has shown that the resolution approach provides a polynomial decision procedure. While this is interesting in itself, a further important aspect is that we can represent a Büchi Automaton (symbolically and directly) as formulae in this fragment, with the emptiness check for such an automaton corresponding to the search for a resolution refutation.

In establishing that some system, Sys, satisfies a property, P, algorithmic, rather than deductive approaches have been predominant. In particular, the model checking approach [22, 2], characterised by checking the emptiness of the automata product[2]

$$\mathcal{A}_{Sys} \times \mathcal{A}_{\neg P}$$

has been very successfully applied.

On the other hand, deductive temporal verification has been largely ignored (though see [17]), often due to its much higher complexity. With our work in this paper, we believe that deductive temporal verification can be successfully applied to such problems, for example by representing $[\![Sys]\!] \wedge \neg P$ in PTL-X$_{\mathcal{A}}$, where $[\![Sys]\!]$ is the temporal/symbolic description/semantics of the behaviour of the system. That PTL-X$_{\mathcal{A}}$ corresponds closely to Büchi Automata which, in turn, are at the heart of algorithmic verification, gives reason for optimism. Thus, our future work concerns developing such a view of deductive temporal verification further, as well as examining more complex (but still tractable) XOR temporal logics [7]. Concerning practical implementation, we note that the complexity given in Section 5 is a *worst case* analysis. With 'clever' implementations, we expect the practical complexity to generally be much lower than this.

Work related to that developed in this paper concerns the excellent analysis by Demri and Schnoebelen [5], work on complexity of fragments of classical logic [19] and our own previous work on the relationship between SNF and Büchi Automata [1] and on other tractable fragments of SNF [8].

[2]Here, $\mathcal{A}_{Sys}$ captures all the paths/executions through Sys, while $\mathcal{A}_{\neg P}$ describes all the paths that satisfy $\neg P$, i.e., all those paths that *do not* satisfy P.

Finally, we would like to thank Radina Yorgova for her work on varieties of RSNF [8] which helped us to formulate the fragment described in this paper.

References

[1] A. Bolotov, M. Fisher, and C. Dixon. On the Relationship between ω-Automata and Temporal Logic Normal Forms. *Journal of Logic and Computation*, 12(4):561–581, August 2002.

[2] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.

[3] A. Degtyarev, M. Fisher, and B. Konev. A Simplified Clausal Resolution Procedure for Propositional Linear-Time Temporal Logic. In U. Egly and C. G. Fermüller, editors, *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-02)*, volume 2381 of *LNCS*, pages 85–99. Springer-Verlag, 2002.

[4] A. Degtyarev, M. Fisher, and B. Konev. Monodic Temporal Resolution. *ACM Transactions on Computational Logic*, 7(1), January 2006.

[5] S. Demri and P. Schnoebelen. The Complexity of Propositional Linear Temporal Logic in Simple Cases. *Information and Computation*, 174(1):84–103, 2002.

[6] C. Dixon. Temporal Resolution using a Breadth-First Search Algorithm. *Annals of Mathematics and Artificial Intelligence*, 22:87–115, 1998.

[7] C. Dixon, M. Fisher, and B. Konev. XOR-Temporal Logics. (Submitted), 2006.

[8] C. Dixon, M. Fisher, and M. Reynolds. Execution and Proof in a Horn-clause Temporal Logic. In *Advances in Temporal Logic*. Kluwer Academic Publishers, 1999.

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

[10] M. Fisher. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4):429–456, August 1997.

[11] M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, January 2001.

[12] 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 (POPL)*, pages 163–173, January 1980.

[13] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, November 2003.

[14] U. Hustadt and B. Konev. TRP++ 2.0: A Temporal Resolution Prover. In *Proceedings of Conference on Automated Deduction (CADE-19)*, number 2741 in LNAI, pages 274–278. Springer, 2003.

[15] U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A Temporal Monodic Prover. In D. Basin and M. Rusinowitch, editors, *Proceedings of the Second International Joint Conference on Automated Reasoning (IJCAR 2004)*, volume 3097 of *LNAI*, pages 326–330. Springer, 2004.

[16] B. Konev, A. Degtyarev, C. Dixon, M. Fisher, and U. Hustadt. Mechanising First-Order Temporal Resolution. *Information and Computation*, 199(1-2):55–86, 2005.

[17] Z. Manna and the STeP group. STeP: Deductive–Algorithmic Verification of Reactive and Real-Time Systems. In *International Conference on Computer Aided Verification (CAV)*, volume 1102 of *LNCS*. Springer-Verlag, 1996.

[18] W. McCune. Otter 2.0. In *Proceedings of Conference on Automated Deduction (CADE-10)*, volume 449 of *LNCS*, pages 663–664, 1990.

[19] T. J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

[20] A. P. Sistla and E. M. Clarke. Complexity of Propositional Linear Temporal Logics. *Journal of the ACM*, 32(3):733–749, July 1985.

[21] A. P. Sistla, M. Vardi, and P. Wolper. The complementation problem for büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[22] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency - Structure versus Automata (Proceedings of 8th Banff Higher Order Workshop)*, volume 1043 of *LNCS*, pages 238–266. Springer, 1996.

[23] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56, 1983.