Events in Linear-Time Properties

Dimitrie O. Păun   Marsha Chechik
Department of Computer Science
University of Toronto
Toronto, ON M5S 3G4, Canada
{dimi,chechik}@cs.toronto.edu

Abstract

For over a decade, researchers in formal methods tried to create formalisms that permit natural specification of systems and allow mathematical reasoning about their correctness. The availability of fully-automated reasoning tools enables more non-specialists to use formal methods effectively — their responsibility reduces to just specifying the model and expressing the desired properties. Thus, it is essential that these properties be represented in a language that is easy to use and sufficiently expressive.

Linear-time temporal logic [21] is a formalism that has been extensively used by researchers for specifying properties of systems. When such properties are closed under stuttering, *i.e. their interpretation is not modified by transitions that leave the system in the same state, verification tools can utilize a partial-order reduction technique [16] to reduce the size of the model and thus analyze larger systems. If LTL formulas do not contain the "next" operator, the formulas are closed under stuttering, but the resulting language is not expressive enough to capture many important properties, e.g., properties involving events. Determining if an arbitrary LTL formula is closed under stuttering is hard — it has been proven to be PSPACE-complete [25].*

In this paper we relax the restriction on LTL that guarantees closure under stuttering, introduce the notion of edges *in the context of LTL, and provide theorems that enable syntactic reasoning about closure under stuttering of LTL formulas.*

1 Introduction

Formal specification of systems has been an active area of research for several decades. From finite-state machines to process algebras to logics, researchers try to create formalisms that would permit natural specification of systems and allow mathematical reasoning about their correctness. However, most of these formalisms have not been adopted widely outside academia — their cost-saving benefits were doubtful, they lacked tool support, and were perceived difficult to apply [27].

Recently, the tools for proving properties of finite-state models are becoming increasingly available and are often used for analyzing requirements, e.g. [2, 3, 10, 4]. These tools typically require the users to specify properties using temporal logics and to describe models of systems using some finite-state transition representation. The tools are based on a variety of verification techniques. For example, SPIN [16] and SMV [22] are based on state-space exploration, also called *model-checking*, Concurrency Workbench [6] on bisimulation, and COSPAN [11] on language containment. Most finite-state verification techniques can be fully automated, and the responsibility of the user reduces to just specifying the model and expressing the desired properties. In this context, it is important that properties can be represented in a language that is easy to use and sufficiently expressive, to enable even fairly novice users to use it effectively.

Linear-time logic (LTL) [21] is a temporal logic that has been extensively used by researchers for specifying properties of systems. A highly desirable property of LTL formulas is that they are *closed under stuttering* [1]. In particular, the mechanical analysis of such formulas, such as by the model-checker SPIN [16], can utilize powerful partial-order reduction algorithms that can dramatically reduce the state-space of the model. Unfortunately, closure under stuttering can be guaranteed only for a subset of LTL [18], and this subset is not expressive enough to represent even fairly simple properties, such as:

> *The magnet of the crane may be deactivated only when the magnet is above the feed belt.*

Users of LTL typically try to remedy this problem by introducing extra variables inside the model — a technique which tends to clutter the model, enlarge the state space, and introduce errors.

Determining whether an LTL formula is closed under stuttering is hard: the problem has been shown to be PSPACE-complete [25]. Even though a complete solution is impractical, we have been able to categorize a subset that is *useful in practice*. In particular, a computationally feasible algorithm which can identify a subclass of closed under stuttering formulas has been proposed in [15] but not yet implemented in SPIN. The algorithm is fairly sophisticated and cannot be applied by hand. Moreover, it is not clear how often the subclass of formulas identified by the algorithm is encountered in practice.

In this paper we relax the restriction on LTL that guarantees closure under stuttering, introduce the notion of *edges* in the context of LTL, and provide theorems that enable syntactic reasoning about closure under stuttering of LTL formulas. The rest of the paper is organized as follows: Section 2 provides some background on LTL and the notation used throughout the paper. Section 3 discusses closure under stuttering and introduces edges. Section 4 gives some important properties of edges and closure under stuttering. Section 5 describes an application of this work to property patterns identified by Dwyer and his colleagues in [8]. We discuss some alternative approaches in Section 6 and conclude the paper in Section 7.

2 Background

We begin by briefly introducing our notation which we have adopted from [12]. A sequence (or string) is a succession of elements joined by semicolons. For example, we write the sequence composed of the first five natural numbers, in order, as $0; 1; 2; 3; 4$ or, more compactly, as $0; ..5$ (note the left-closed, right-open interval). We can obtain an item of the sequence by subscripting: $(0; 2; 4; 5)_2 = 4$. When the subscript is a sequence, we obtain a subsequence: $(0; 3; 1; 5; 8; 2)_{1;2;3} = (0; 3; 1; 5; 8; 2)_{1;..4} = 3; 1; 5$.

A state is modeled by a function that maps variables to their values, so the value of variable a in state s_0 is $s_0(a)$. We denote the set of all infinite sequences of states as $\mathbf{St}^\infty$, and the set of natural numbers as $\mathbb{N}$.

Boolean expressions are connected by $\neg$(not), $\wedge$ (and), $\vee$ (or), $\Rightarrow$ (implies), $\Leftarrow$ (is implied by), and $=$ (if and only if). To reduce the clutter of parenthesis, we denote the main connective in a formula by a bigger and bolder symbol, e.g. $\boldsymbol{=}$. We consider the connectives to have decreasing precedence as follows: $=$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$; the connectives $\boldsymbol{=}$, $\boldsymbol{\Rightarrow}$, and $\boldsymbol{\Leftarrow}$ have the lowest precedence. For example, the formula $a \wedge b \vee b \boldsymbol{=} a \Rightarrow b$ should be parsed as: $((a \wedge b) \vee b) = (a \Rightarrow b)$.

Linear time temporal logic (LTL) is a language for describing and reasoning about sequences of states. These sequences can be interpreted in a variety of ways: the state of the world as it evolves over time, the state of a program as it is executing, and so forth. Informally, LTL is comprised of propositional formulas and temporal connectives $\Box$ (always), $\Diamond$ (eventually), $\circ$ (next), and $\mathcal{U}$ (until). The first three operators are unary, while the last one is binary. Using these operators, one can express properties about the evolution of a system. For example, a formula $p \vee \circ q$ indicates that either p holds in the starting state of the system, or q holds in the following state. A formula $\Box(p \Rightarrow \Diamond q)$ indicates that each occurrence of p is followed, at some point, by an occurrence of q. We now define the formal semantics of an LTL formula, based on its syntactic structure. Let A and B be LTL formulas, let a be a variable name, and let s be a sequence of states, i.e. $s \in \mathbf{St}^\infty$. We denote an interpretation of formula F in state sequence s as $s[\![F]\!]$, and define it as follows:

$$
\begin{aligned}
s[\![a]\!] &= s_0(a) \\
s[\![\neg A]\!] &= \neg s[\![A]\!] \\
s[\![A \wedge B]\!] &= s[\![A]\!] \wedge s[\![B]\!] \\
s[\![\circ A]\!] &= s_{1;..\infty}[\![A]\!] \\
s[\![\Box A]\!] &= \forall i \in \mathbb{N} \cdot s_{i;..\infty}[\![A]\!] \\
s[\![\Diamond A]\!] &= \exists i \in \mathbb{N} \cdot s_{i;..\infty}[\![A]\!] \\
s[\![A\, \mathcal{U}\, B]\!] &= \exists i \in \mathbb{N} \cdot (s_{i;..\infty}[\![B]\!] \wedge \\
&\qquad \forall j \in \{0, 1, .., i-1\} \cdot s_{j;..\infty}[\![A]\!])
\end{aligned}
$$

For example, $s[\![\Diamond A]\!]$ means that a formula A must hold at some point i in the sequence s. We do not describe other boolean operators as they can be expressed in terms of negation and conjunction.

We say that an LTL formula F is closed under stuttering when its interpretation remains the same under state sequences that differ only by repeated states. We denote a closed under stuttering formula as $\ll F \gg$, and formally define it as follows:

Definition 1 $\ll F \gg \boldsymbol{=}$

$$\forall s \in \mathbf{St}^\infty \cdot \forall i \in \mathbb{N} \cdot s[\![F]\!] = (s_{0;..i}; s_i; s_{i;..\infty})[\![F]\!]$$

In other words, given any state sequence s, we can repeat any of its states without changing the interpretation of F. Note that $s_{0;..i}; s_i; s_{i;..\infty}$ is a sequence of states that differ from s only by the repeated state s_i.

It is easy to see that any LTL formula that does not contain the "$\circ$" operator is closed under stuttering.

For example, $\Box a$ is closed under stuttering because no matter how much we repeat states, we cannot change the value of a. On the other hand, the formula $\circ a$ is not closed under stuttering. We can see that by considering the state sequence s in which $s_0(a)$ is true and $s_1(a)$ is false. Then $\circ a$ is false when we evaluate it in s, and true when we evaluate it in $s_0; s$.

3 Closure Under Stuttering and Edges

Our aim is to capture a large set of formulas that use the "$\circ$" operator but are still closed under stuttering. We begin by developing the intuition behind the main theorem that enables us to generate this class of formulas. Let B be an LTL formula that is closed under stuttering. Consider the formula $\circ B$: it may not be closed under stuttering; furthermore, we can not correct this by simply quantifying $\circ B$ with '$\Box$' or '$\Diamond$'. To understand how stuttering can modify the interpretation of $\circ B$, we illustrate two different stuttering scenarios in Figure 1, where s is a sequence of states, and s' and s'' are derived from s by stuttering the fifth and the first states respectively. Note that the type of stuttering exemplified in s', that is, repeating any part of the sequence with the exception of the first state, causes no harm because B itself is closed under stuttering. However, stuttering the first state, as shown in s'', may cause problems, because $\circ B$ is evaluated on a state sequence that includes the new state.

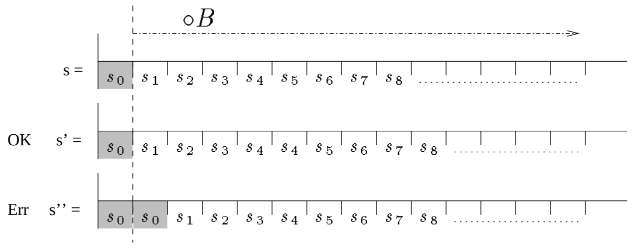

Figure 1: Effects of stuttering on formula $\circ B$.

To circumvent this problem, we conjoin $\circ B$ with another formula, F, and quantify the resulting expression with '$\Diamond$': $\Diamond(F \wedge \circ B)$. We need the quantification to "follow" the state s_1, as s_0 can be stuttered any number of times, while the conjunct F is needed to ignore the leading sequence of repeated states s_0.[1] A possible way to determine when a state is repeated is to check that an LTL formula A that is closed under stuttering does not change its interpretation from the current to the next state. Therefore, we can define F as $\neg A \wedge \circ A$. Note that F is guaranteed to be false only when the sequence starts with a repeated state.

We now claim that the formula developed above is closed under stuttering:

Theorem 1

$$\ll A \gg \wedge \ll B \gg \Rightarrow \ll \Diamond(\neg A \wedge \circ A \wedge \circ B) \gg$$

Proof Sketch: Let $s, s' \in \mathbf{St}^\infty$, such that s' is the same as s except that the first state of s' is not stuttered. In other words, we can repeat the first state of s' a number of times and obtain s. We can rewrite the formula $\Diamond(\neg A \wedge \circ A \wedge \circ B)$ as $\Diamond(\neg A \wedge \circ G)$, where $G = A \wedge B$. If we let $F = \Diamond(\neg A \wedge \circ G)$, we can see that $s[\![F]\!] = s'[\![F]\!]$. So, the interpretation of F is not affected by stuttering except perhaps when we stutter the first state s_0. We have argued in the discussion leading to the theorem that this is the case provided that A and G are closed under stuttering, which is true here. Thus, we only need to worry about state sequences s which have the first state stuttered. However, in such cases the expression $\neg A \wedge \circ A$ becomes false because A is closed under stuttering. This means that the entire preamble of stuttered states that might change the interpretation of the formula is in fact ignored, and thus the formula remains closed under stuttering. ∎

This theorem is the main result that allows us to build formulas that contain the "$\circ$" operator and are still closed under stuttering. The theorem was enabled by the formula $\neg A \wedge \circ A$, which expresses a *change* in A. More exactly, it expresses an *up edge* in A. We denote an up edge by $\uparrow$, a down edge by $\downarrow$, and an up or down edge by $\updownarrow$. Their formal definitions follow.

Definition 2 *If A is an LTL formula, then*

$\uparrow A$	$=$	$\neg A \wedge \circ A$	—	*up or rising edge*
$\downarrow A$	$=$	$A \wedge \circ \neg A$	—	*down or falling edge*
$\updownarrow A$	$=$	$\uparrow A \vee \downarrow A$	—	*any edge*

The edges are also called *events*, as they capture the same notion as the events proposed by the Software Cost Reduction (SCR) researchers [14, 13]. The SCR events cannot explicitly include temporal operators, whereas our formalism enables reasoning about events in arbitrary LTL formulas. For example, a formula $\uparrow\Box A$ has a well-defined interpretation in our language. We also note here a strong analogy between our (logical) edges and signal edges. Computer engineers have made good use of edges in electrical signals — most circuitry is driven by edges, e.g. the clock. Edges are

[1] Note that we could not have used $\Box$ for quantification as it distributes over conjunction. The choice of a quantifier is dictated by the boolean operator between F and $\circ B$.

also widely used in other engineering disciplines, e.g. electrical engineering and telecommunications. It is surprising that we managed to work around them for so long in the model checking world!

4 Properties

In this section we present a few important properties of edges and closure under stuttering. We begin by noting that edges are related by the following formulas:

$$\begin{aligned} \uparrow\neg A &= \downarrow A && (1)\\ \downarrow\neg A &= \uparrow A && (2)\\ \updownarrow\neg A &= \updownarrow A && (3) \end{aligned}$$

These formulas allow us to switch between the different types of edges easily. The following are some general properties of closure under stuttering:

$$\begin{aligned} a \text{ is a variable} &\Rightarrow \ll a \gg && (4)\\ \ll A \gg &= \ll \neg A \gg && (5)\\ \ll A \gg \wedge \ll B \gg &\Rightarrow \ll A \wedge B \gg && (6)\\ \ll A \gg &\Rightarrow \ll \Box A \gg && (7)\\ \ll A \gg &\Rightarrow \ll \Diamond A \gg && (8)\\ \ll A \gg \wedge \ll B \gg &\Rightarrow \ll A \,\mathcal{U}\, B \gg && (9) \end{aligned}$$

Note that property (5) is an equality indicating that when a formula is negated, its closure under stuttering property is preserved. Property (6) indicates that if two formulas are closed under stuttering, then so is their conjunction. These two formulas allow us to conclude that

$$\ll A \gg \wedge \ll B \gg \Rightarrow \ll A * B \gg$$

where $*$ is any of $\wedge$, $\vee$, $\Rightarrow$, $\Leftarrow$, or $=$. Such properties enable reasoning about closure under stuttering of a formula by looking at its components. Finally, formula (5) together with (1) and (2) allows the interchangeable use of $\downarrow$ and $\uparrow$ when analyzing properties of the form

$$\ll A \gg \Rightarrow f(\uparrow A)$$

Thus, in the rest of the paper we talk only about the $\uparrow$-edges in the context of closure under stuttering.

Below we discuss closure under stuttering properties that contain edges and the "$\circ$" operator. The first property is a corollary of Theorem 1:

Property 1

$$\ll A \gg \wedge \ll B \gg \wedge \ll C \gg \Rightarrow \ll \Diamond(\uparrow A \wedge \circ B \wedge C) \gg$$

It has two simplified versions:

$$\ll A \gg \wedge \ll B \gg \Rightarrow \ll \Diamond(\uparrow A \wedge B) \gg$$

and

$$\ll A \gg \wedge \ll B \gg \Rightarrow \ll \Diamond(\uparrow A \wedge \circ B) \gg$$

that we often encountered in practice. In both cases an *existence* property is formalized. The formulas say that the event $\uparrow A$ must happen and then B holds. In these versions, B is evaluated right before or right after the event, respectively.

Property 2

$$\ll A \gg \wedge \ll B \gg \wedge \ll C \gg \Rightarrow \ll \Box(\uparrow A \Rightarrow \circ B \vee C) \gg$$

This property is logically equivalent to Property 1. Its two simplified versions are

$$\ll A \gg \wedge \ll B \gg \Rightarrow \ll \Box(\uparrow A \Rightarrow B) \gg$$

and

$$\ll A \gg \wedge \ll B \gg \Rightarrow \ll \Box(\uparrow A \Rightarrow \circ B) \gg$$

They express a *universality* property: whenever the event $\uparrow A$ happens, B will hold. As in the case of Property 1, B is evaluated right before or right after the event, respectively.

Consider the following example:

> *Items should only be dropped on the table.*

This property can be formalized as

$$\Box(\downarrow hold \Rightarrow pos = above\_tbl),$$

where *hold* is a state variable that is true when we hold an item, and *pos* is a state variable indicating the position. Note that an item is considered dropped if we hold it in one state and do not hold it in the next. Formally, we express "dropped" as $hold \wedge \circ\neg hold$ or as $\downarrow hold$.

The last property deals with the "until" operator:

Property 3

$$\ll A \gg \wedge \ll B \gg \wedge \ll C \gg \wedge \ll D \gg \wedge \ll E \gg \wedge \ll F \gg$$
$$\Rightarrow \ll (\neg\uparrow A \vee \circ B \vee C) \,\mathcal{U}\, (\uparrow D \wedge \circ E \wedge F) \gg$$

There are many simplified expressions for this property which are omitted here for brevity.

For example, a property

> *Initially, no items should be dropped on the table before the operator pushes and releases the GO button.*

can be formalized as

$$\neg\downarrow hold\ \mathcal{U} \downarrow button,$$

where *hold* has the same meaning as before, and *button* is a state variable which is true when the button is pressed and false otherwise.

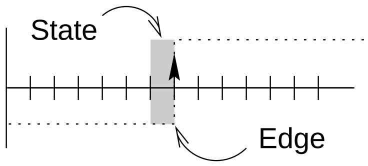

Figure 2: Edge-detecting state

In order to effectively express properties containing edges, it is important to realize that an edge is detected just *before* it occurs, as illustrated in Figure 2.[2] That is, $\uparrow A$ becomes true in the state where A is false. For example, consider the following property:

> *After the robot deposits an object on the belt, it should not hold another object until the sensor at the end of the belt is true.*

If we formalize it as

$$\Box(\downarrow hold \Rightarrow (\neg hold\ \mathcal{U}\ sensor)),$$

we do not get the correct formula since its consequent would be evaluated one state too early: $\downarrow hold$ is detected when *hold* is true, but requires *hold* to remain false until *sensor* is true. The formula can be fixed by considering the consequent in the "next" state:

$$\Box(\downarrow hold \Rightarrow \circ(\neg hold\ \mathcal{U}\ sensor))$$

5 Edges and Patterns

A pattern-based approach to the presentation, codification and reuse of property specifications for finite-state verification was proposed by Dwyer and his colleagues in [9, 8]. They performed a large-scale study in which specifications containing over 500 temporal properties were collected and analyzed. They noticed that over 90% of these could be classified under one of the proposed patterns [9].

We discuss two directions for integrating our work into the pattern system: extending the system to include events based on edges, and evaluating the effectiveness of our theorems in determining closure under stuttering for the newly created, edge-based formulas. In the rest of the section we briefly discuss the property pattern system (following the presentation in [8]), describe our extensions based on the usage of edges, and conclude with discussing closure under stuttering.

[2] Note that we can move the detection of the edge after it occurs if we replace the "next" by the "previous" operator.

5.1 The Pattern System

The patterns enable non-experts to read and write formal specifications for realistic systems and facilitate easy conversion of specifications between formalisms. Currently, the properties can be expressed in a variety of formalisms such as LTL, computational tree logic (CTL) [5], quantified regular expressions (QRE) [23], and other state-based and event-based formalisms.

The patterns are organized in a hierarchy based on their semantics, as illustrated in Figure 3. Some of the patterns are described below:

Absence A condition does not occur within a scope;

Existence A condition must occur within a scope;

Universality A condition occurs throughout a scope;

Response A condition must always be followed by another within a scope;

Precedence A condition must always be preceded by another within a scope.

Each pattern is associated with several *scopes* — the regions of interest over which the condition is evaluated. There are five basic kinds of scopes:

A. Global The entire state sequence;

B. Before R The state sequence up to condition R;

C. After Q The state sequence after condition Q;

D. Between Q **and** R The part of the state sequence between condition Q and condition R;

E. After Q **Until** R Similar to the previous one except that the designated part of the state sequence continues even if the second condition does not occur.

These scopes are depicted in Figure 4. The scopes were initially defined in [9] to be closed-left, open-right intervals, although it is also possible to define other combinations, such as open-left, closed-right intervals.

For example, an LTL formulation of the property "S precedes P between Q and R" (**Precedence** pattern with "between Q and R" scope) is:

$$\Box((Q \wedge \Diamond R) \Rightarrow (\neg P\ \mathcal{U}\ (S \vee R)))$$

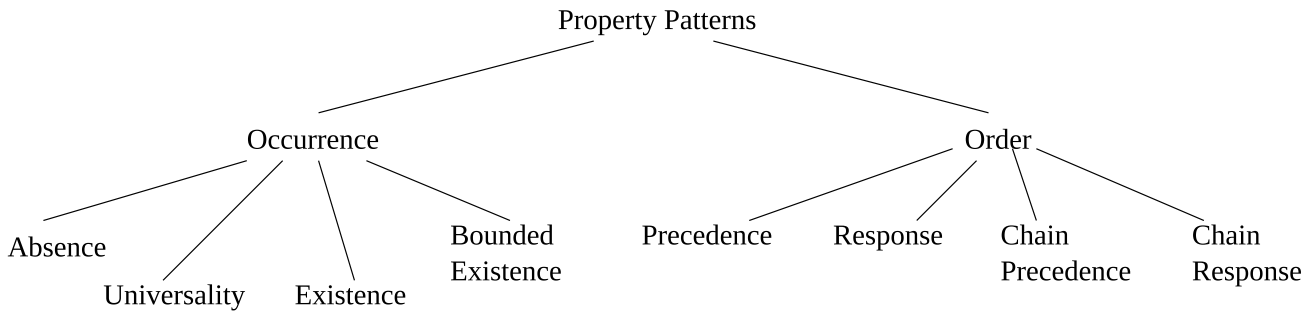

Figure 3: A Pattern Hierarchy.

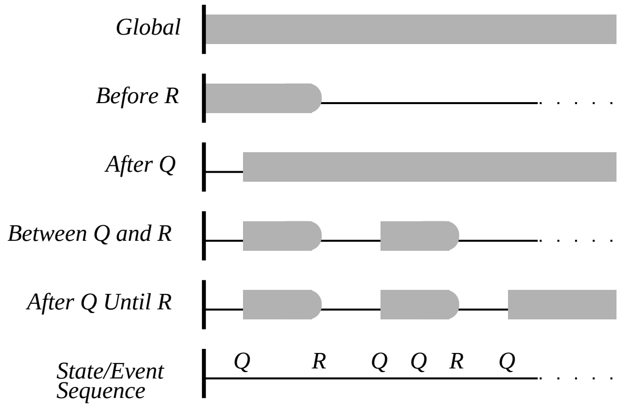

Figure 4: Pattern Scopes.

Even though the pattern system is formalism-independent [8], we are only interested in the way the patterns are expressed in LTL.

5.2 Events in LTL Patterns

It is often natural to express properties using changes of states — edges. As the original pattern system was state-based, we tried to extend it by incorporating edge-based events. These events can be used for specifying the conditions as well as for defining the bounding scopes. For example, we may want to express the properties "Event S precedes event P between states Q and R", or "State S precedes state P between events Q and R". These properties are often hard to specify correctly, and since all of them contain the "$\circ$" operator (either directly or indirectly through the use of edges), it is not trivial to determine if they are closed under stuttering.

Introducing edges into the patterns generates an explosion in the number of formulas. The patterns contain up to two conditions, while the bounding interval has up to two ends; each of these can be either state-based or edge-based, giving us up to 16 different combinations. Moreover, when a condition and an interval end are of the same type (either state-based or edge-based), we can choose to make the interval open or closed, leading to even more possibilities. Note that when the condition and the interval end are of different types, there is no ambiguity because edges occur *between* states.

Due to the large number of possible formulas, we extended only some of the patterns: **Absence**, **Existence**, **Universality**, **Precedence**, and **Response**. In the original pattern system, the conditions were represented by formulas P and S, while the bounding interval was defined by formulas Q and R. We have considered the following possibilities:

0. P, S — states, Q, R — states;
1. P, S — states, Q, R — up edges;
2. P, S — up edges, Q, R — states;
3. P, S — up edges, Q, R — up edges.

Combination 0 corresponds to the original formulation of [8], where all of P, S, Q and R are state-based. The remaining three combinations are our extensions of the pattern system. We assume that multiple events can happen simultaneously, but only consider closed-left, open-right intervals, as in the original system. We note, however, that it is perfectly possible to have formulas for all other combinations of interval bounds. Down edges can be substituted for up edges without changing the formulas.

In Figure 5 we show the LTL formulations for the **Existence** pattern. Each of the scopes is associated with four formulas corresponding to the four combinations of state-based and edge-based conditions and interval bounds we have considered. We have modified several of the 0-formulas (i.e. state-based conditions and intervals) from their original formulations of [8] to remove assumptions of interleaving and make them consistent with the left-closed, right-open intervals. For brevity, this is the only pattern that we detail in this paper. We are currently working on integrating our work into the pattern system of Dwyer and his colleagues. Meanwhile, the complete set of formulas we developed is available at `http://www.cs.toronto.edu/~chechik/edges.html`.

5.3 Closure Under Stuttering and Patterns

Our primary concern associated with the edge-based extension of the pattern system is to analyze the newly-created formulas for the closure under stuttering property. Our interest in this problem is twofold: first, we know that edge-based formulas can have practical relevance only if they are closed under stuttering, and second, these formulas provide a good test-bed for the closure under stuttering theorems we have developed.

Let us consider an example: a robot must pick up a metal blank from a feed belt, weigh it, and deposit it in a press. The specification of the robot says:

> *The robot must weigh the blank after pickup from the feedbelt, but before the deposit in the press.*

The robot is equipped with a magnet and a scale at the end of its arm. The status of the magnet is reported through a state variable *mgn* which is true when the magnet is on, while the scale reports a successful weighing when the state variable *scl* is true.

Clearly, this fits the "Existence" pattern: a state base condition (weighing) must happen between two events (pickup and deposit). In Figure 5, we can find the desired formula as D.1. Using $\uparrow mgn$ and $\downarrow mgn$ to model the pickup and the deposit event, respectively, and plugging the events into the template yields the formula:

$$\Box(\uparrow mgn \wedge \Diamond\downarrow mgn \Rightarrow \circ(\neg\downarrow mgn\, \mathcal{U}\, scl) \wedge \neg\downarrow mgn)$$

In order to prove that this property is closed under stuttering, we need to show that if components of the template, Q, R and P are closed under stuttering, then so is the template. Q, R and P are just variables, trivially closed under stuttering, and the analysis of the template appears below:

$$
\begin{array}{cl}
 & \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \circ(\neg\uparrow R\, \mathcal{U}\, P) \wedge \neg\uparrow R)\gg \\
 & \text{by the laws of logic and LTL} \\
= & \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \circ(\neg\uparrow R\, \mathcal{U}\, P))\wedge \\
 & \quad \Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \neg\uparrow R)\gg \\
 & \text{by 6} \\
\Leftarrow & \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \circ(\neg\uparrow R\, \mathcal{U}\, P))\gg\wedge \\
 & \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \neg\uparrow R)\gg \\
 & \text{by Property 2, we get:} \\
\Leftarrow & \ll Q\gg \wedge \ll\Diamond\uparrow R\gg \wedge \ll\neg\uparrow R\, \mathcal{U}\, P\gg\wedge \\
 & \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \neg\uparrow R)\gg \\
 & \text{by Property 1, this simplifies to:} \\
\Leftarrow & \ll Q\gg \wedge \ll R\gg \wedge \ll\neg\uparrow R\, \mathcal{U}\, P\gg\wedge \\
 & \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \neg\uparrow R)\gg \\
 & \text{by Property 3, we get:} \\
\Leftarrow & \ll Q\gg \wedge \ll R\gg \wedge \ll R\gg \wedge \ll P\gg\wedge \\
 & \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \neg\uparrow R)\gg \\
 & \text{by the rules of logic we get:} \\
= & \ll Q\gg \wedge \ll R\gg \wedge \ll P\gg\wedge \\
 & \ll\neg\Diamond(\uparrow Q \wedge \Diamond\uparrow R \wedge \uparrow R)\gg \\
 & \text{by 5 and Property 1:} \\
\Leftarrow & \ll Q\gg \wedge \ll R\gg \wedge \ll P\gg\wedge \\
 & \ll Q\gg \wedge \ll R\gg \wedge \ll\Diamond\uparrow R\gg \\
 & \text{by Property 1 again:} \\
\Leftarrow & \ll Q\gg \wedge \ll R\gg \wedge \ll P\gg\wedge \\
 & \ll Q\gg \wedge \ll R\gg \wedge \ll R\gg
\end{array}
$$

Thus, we have proven that

$$\ll P\gg \wedge \ll Q\gg \wedge \ll R\gg \Rightarrow \ll\Box(\uparrow Q \wedge \Diamond\uparrow R \Rightarrow \circ(\neg\uparrow R\, \mathcal{U}\, P) \wedge \neg\uparrow R)\gg$$

which is exactly the desired property.

Note that, although the property is fairly complicated, the proof is not long, is completely syntactic, and each step in it is easy. Similar proofs were found for all of the new edge-based formulas [24]. Such proofs can potentially be performed by a theorem-prover like PVS [28] with little guidance from the user. We are currently investigating the feasibility of doing so.

6 Discussion and Related Work

Before writing this paper, we searched through numerous research publications and web sites, looking for good examples of LTL formulas containing the "next" state, but surprisingly found just a few. We also looked at over 500 temporal formulas collected by Dwyer and his colleagues [7, 9] and found virtually no

A. P Exists Globally	B. P Exists Before R	C. P Exists After Q
0. $\Diamond P$	0. $\Diamond R \Rightarrow \neg(\neg P\,\mathcal{U}\,R)$	0. $\Diamond Q \Rightarrow \Diamond(Q \wedge \Diamond P)$
1. $\Diamond P$	1. $\Diamond{\uparrow}R \Rightarrow (\neg{\uparrow}R\,\mathcal{U}\,P)$	1. $\Diamond{\uparrow}Q \Rightarrow \Diamond({\uparrow}Q \wedge \circ\Diamond P)$
2. $\Diamond{\uparrow}P$	2. $\Diamond R \Rightarrow \neg(\neg{\uparrow}P\,\mathcal{U}\,R)$	2. $\Diamond Q \Rightarrow \Diamond(Q \wedge \Diamond{\uparrow}P)$
3. $\Diamond{\uparrow}P$	3. $\Diamond{\uparrow}R \Rightarrow \neg(\neg{\uparrow}P\,\mathcal{U}\,{\uparrow}R)$	3. $\Diamond{\uparrow}Q \Rightarrow \Diamond({\uparrow}Q \wedge \Diamond{\uparrow}P)$

D. P Exists Between Q and R	E. P Exists After Q Until R
0. $\Box(Q \wedge \Diamond R \Rightarrow \neg(\neg P\,\mathcal{U}\,R) \wedge \neg R)$	0. $\Box(Q \Rightarrow \textbf{if } \Diamond R \textbf{ then } \neg(\neg P\,\mathcal{U}\,R) \wedge \neg R \textbf{ else } \Diamond P)$
1. $\Box({\uparrow}Q \wedge \Diamond{\uparrow}R \Rightarrow \circ(\neg{\uparrow}R\,\mathcal{U}\,P) \wedge \neg{\uparrow}R)$	1. $\Box({\uparrow}Q \Rightarrow \circ(\neg{\uparrow}R\,\mathcal{U}\,P) \wedge \neg{\uparrow}R)$
2. $\Box(Q \wedge \Diamond R \Rightarrow \neg(\neg{\uparrow}P\,\mathcal{U}\,R) \wedge \neg R)$	2. $\Box(Q \Rightarrow \textbf{if } \Diamond R \textbf{ then } \neg(\neg{\uparrow}P\,\mathcal{U}\,R) \wedge \neg R \textbf{ else } \Diamond{\uparrow}P$
3. $\Box({\uparrow}Q \wedge \Diamond{\uparrow}R \Rightarrow \neg(\neg{\uparrow}P\,\mathcal{U}\,{\uparrow}R) \wedge \neg{\uparrow}R)$	3. $\Box({\uparrow}Q \Rightarrow \textbf{if } \Diamond{\uparrow}R \textbf{ then } \neg(\neg{\uparrow}P\,\mathcal{U}\,{\uparrow}R) \wedge \neg{\uparrow}R \textbf{ else } \Diamond{\uparrow}P)$

Figure 5: Formulations of the **Existence** Pattern

explicit usage of events or the "next" operator. This led us to conclude that the community is almost religiously avoiding the "next" state operator, replacing it with a variety of surrogates, most of which are neither elegant nor expressive.

For example, to simulate an up edge, it is customary to create an extra variable, $\hat{a}$, that signals a change in a by being temporarily true when a is changed. This can be modeled by a concurrent assignment to a and $\hat{a}$:

$$\textbf{atomic}\{\hat{a} \leftarrow 1; a \leftarrow 1;\}\ \hat{a} \leftarrow 0;$$

Note that $\hat{a}$ is weaker than ${\uparrow}a$ since in systems containing more than one process, $\hat{a}$ is true for a longer time than ${\uparrow}a$. Since ${\uparrow}a \Rightarrow \hat{a}$, replacing ${\uparrow}a$ by $\hat{a}$ in LTL properties can lead to hard-to-interpret verification results. For example, a property

$$\Box(\hat{a} \Rightarrow \Diamond p)$$

is a *conservative* approximation of

$$\Box({\uparrow}a \Rightarrow \circ\Diamond p)$$

That is, if the former property holds, so does the later, but the converse is not true. In many such cases the approximation is too strong, forcing the user to modify a possibly correct model just to be able to verify the property. Such approximations can be so strong that they are always false in most models. For example, performing the substitution in the formula:

$$\Box({\uparrow}a \Rightarrow \circ(\neg{\uparrow}a\,\mathcal{U}\,{\uparrow}b))$$

will most likely result in a formula that is always false because $\hat{a}$ can be true for several consecutive states. A property

$$\Diamond(\hat{a} \wedge \hat{b})$$

is an *optimistic* approximation of

$$\Diamond({\uparrow}a \wedge {\uparrow}b)$$

That is, if the former property does not hold, neither does the later, but the converse is not true. This means that the approximation cannot be used for checking the validity of the model. Combining conservative and optimistic approximations can void the resulting formula of any meaning.

The users of the model-checker SPIN [16] probably constitute the largest LTL user-group[3]. However, judging from the four years of proceedings of the SPIN workshop, available at [20], they seldom if ever use properties involving events because of the common misconception that no formulas containing "next" are closed under stuttering, expressed, e.g., by Kamel and Leue in [17]. This is, in our opinion, a serious problem because we believe that edges are required to express most nontrivial properties. For example, during our work [26] on the Production Cell [19], edges were required in 10 out of 14 properties that we formalized, and in many of these, simulating edges by introducing extra variables was not possible.

A restricted use of "next", similar to ours, is also advocated by Lamport in his Temporal Language of Actions (TLA) [18], where "next" is replaced by "primed variables", e.g., a' indicates the value of a in the next state. However, this is not sufficient to guarantee closure under stuttering and an additional restriction is placed on the TLA formulas. This restriction is similar in form to the one imposed by Theorem 1.

[3] SPIN has over 4000 installations world-wide.

7 Summary and Conclusion

The "next" operator in linear-time temporal logics is required for reasoning about events. However, it is seldom if ever used in practice because of a false belief that it does not allow construction of formulas that are closed under stuttering. Instead, people introduce extra variables to simulate events. These variables clutter the model and make it harder to analyze. Moreover, results of the verification with respect to these properties often cannot be interpreted correctly without complete understanding of the modeling language and logic, leading to errors among novice and even expert users.

In this paper, we have introduced the notion of edges in the context of LTL, a concept that allows us to easily express temporal properties involving events. We have also provided a number of theorems that enable syntax-based analysis of a large class of formulas for closure under stuttering. These theorems can be easily added to a theorem prover for mechanized checking. In addition, we extended the patterns identified in [8] with event-based formulations, and proved that the resulting formulas are closed under stuttering using the theorems presented in this paper. Unfortunately, unlike the "next"-free LTL, our language of edges is not closed, i.e., it is possible to use this language to state a property which is not closed under stuttering. However, we feel that it can express and enable analysis of the majority of formulas that are encountered in practice. For example, we were easily able to check the formalization of properties of the Production Cell System.

We hope that the work presented in this paper will contribute to increasing the usability of formal methods, at least in the linear-temporal logic domain.

Acknowledgments

We would like to thank Rick Hehner, Connie Heitmeyer, and the anonymous referees for their helpful comments on the earlier drafts of this paper. The research was supported in part by the Natural Science and Engineering Research Council of Canada.

References

[1] M. Abadi and L. Lamport. "The Existence of Refinement Mappings". *Theoretical Computer Science*, 82(2):253–284, 1991.

[2] J.M. Atlee and J. Gannon. "State-Based Model Checking of Event-Driven System Requirements". *IEEE Transactions on Software Engineering*, pages 22–40, January 1993.

[3] Ramesh Bharadwaj and Connie Heitmeyer. "Model Checking Complete Requirements Specifications Using Abstraction". *Journal of Automated Software Engineering*, 6(1), January 1999.

[4] M. Chechik. "SC(R)^3: Towards Usability of Formal Methods". In *Proceedings of CASCON'98*, pages 177–191, November 1998.

[5] E.M. Clarke, E.A. Emerson, and A.P. Sistla. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[6] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. "The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems". *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[7] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "A System of Specification Patterns". Patterns catalog is available at `http://www.cis.ksu.edu/\verb1~1dwyer/spec-patterns.html`. A collection of over 500 temporal properties is available at `http://www.cis.ksu.edu/\verb1~1dwyer/SPAT/SURVEY/ALL.raw`, 1998.

[8] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "Property Specification Patterns for Finite-state Verification". In *Proceedings of 2nd Workshop on Formal Methods in Software Practice*, March 1998.

[9] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "Patterns in Property Specifications for Finite-State Verification". In *Proceedings of 21st International Conference on Software Engineering*, May 1999.

[10] Steve Easterbrook, Robyn Lutz, Richard Covington, John Kelly, Yoko Ampo, and David Hamilton. "Experience Using Lightweight Formal Methods for Requirements Modeling". *IEEE Transactions on Software Engineering*, 24(1):4–14, January 1998.

[11] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. "STATEMATE: A Working Environment for the Development of Complex Reactive Systems". *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[12] Eric C. R. Hehner. *A Practical Theory of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, New York, 1993.

[13] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. "Automated Consistency Checking of Requirements Specifications". *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, July 1996.

[14] K. Heninger. "Specifying Software Requirements for Complex Systems: New Techniques and Their Applications". *IEEE Transactions on Software Engineering*, SE-6(1):2–12, January 1980.

[15] G. J. Holzmann and O. Kupferman. "Not Checking for Closure under Stuttering". In *Proceedings of SPIN'96*, 1996.

[16] G.J. Holzmann. "The Model Checker SPIN". *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[17] Moataz Kamel and Stefan Leue. "Validation of Remote Object Invocation and Object Migration in CORBA GIOP using Promela/Spin". In *Proceedings of SPIN'98*, Paris, France, November 1998.

[18] Leslie Lamport. "The Temporal Logic of Actions". *ACM Transactions on Programming Languages and Systems*, 16:872–923, May 1994.

[19] Claus Lewerentz and Thomas Lindner, editors. *Formal Development of Reactive Systems. Case Study Production Cell*. Springer-Verlag, Berlin, 1995.

[20] Gerarld Holzmann (maintainer). "On the Fly, LTL Model-checking with SPIN". Description of the model-checker and pointers to proceedings of SPIN workshops are available at `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[21] Z. Manna and A. Pnueli. "The Anchored Version of the Temporal Framework". In W.P. de Roever J.W. de Bakker and G. Rozenburg, editors, *Models of Concurrency: Linear, Branching and Partial Orders*, LNCS. Springer-Verlag, 1989.

[22] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[23] Kurt M. Olender and Leon J. Osterweil. "Cecil: A Sequencing Constraint Language for Automatic Static Analysis Generation". *IEEE Transactions on Software Engineering*, 16(3):268–280, March 1990.

[24] Dimi Paun. Closure under stuttering in temporal formulas. Master's thesis, Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3G4, CANADA, 1999. April.

[25] Doron Peled, Thomas Wilke, and Pierre Wolper. "An Algorithmic Approach for Checking Closure Properties of ω-Regular Languages". In *Proceedings of CONCUR '96: 7th International Conference on Concurrency Theory*, August 1996.

[26] Dimitrie O. Păun, Marsha Chechik, and Bernd Biechelle. "Production Cell Revisited". In *Proceedings of SPIN'98*, November 1998.

[27] D. Rosenblum. "Formal Methods and Testing: Why the State-of-the-Art is not the State-of-the-Practice?". *ACM SIGSOFT Software Engineering Notes*, 21(4), July 1996. (ISSTA'96/FMSP'96 panel summary).

[28] N. Shankar, S. Owre, and J. Rushby. "The PVS Proof Checker: A Reference Manual (Beta Release)". Technical report, Computer Science Lab, SRI International, Menlo Park, CA, March 1993.