Verification of reactive systems using temporal logic with clocks

Chuchang Liu a,*, Mehmet A. Orgun b

a Information Technology Division, Defence Science and Technology Organisation, PO Box 1500, Salisbury, SA 5108, Australia
b Department of Computing, Macquarie University, NSW 2109, Australia

Abstract

This paper presents a framework for the specification and verification of timing properties of reactive systems using Temporal Logic with Clocks (TLC). Reactive systems usually contain a number of parallel processes, therefore, it is essential to study and analyse each process based on its own local time. TLC is a temporal logic extended with multiple clocks, and it is in particular suitable for the specification of reactive systems. In our framework, the behavior of a reactive system is described through a formal specification; its timing properties, including safety and liveness properties, are expressed by TLC formulas. We also propose several demonstration techniques, such as an application of local reasoning and deriving fixed-time rules from the proof system of TLC, for proving that a reactive system meets its temporal specification. Under the proposed framework, the timing properties of a reactive system can therefore be directly reasoned about from the formal specification of the system. © 1999 Elsevier Science B.V. All rights reserved.

1. Introduction

Temporal logic has been widely used as a formalism for program specification and verification [8, 20, 29, 31], temporal reasoning [41, 42], modeling temporal databases [3, 14, 15, 37], simulation applications [26, 43] and so on. More recently, there has been a substantial interest in the use of temporal logic for specifying timing properties of reactive systems [5, 6, 12, 21, 31–33]. This paper presents a framework for the specification and verification of timing properties of reactive systems using Temporal Logic with Clocks (TLC).

When considering the execution of conventional programs, we find that such an execution can be viewed as consisting of three consecutive activities: the environment prepares an input, the program performs its computation until it terminates, and the
environment uses the output generated by the program. Such a program can usually be specified by characterizing the relation between its initial and final states. For such specifications, it is usually sufficient to use first-order logic to provide an adequate formulation and reasoning tool. However, in reactive programming, if a program is not fast enough, it may miss some deadlines or fail to respond to some important events. For a reactive system, it is important to describe the situation in which the program and its environment act concurrently, rather than sequentially. Reactive systems usually contain several parallel processes, which are running concurrently. From the point of view of each process, the rest of the program can be viewed as an environment that continuously interacts with the process. Such a system can be effectively specified in terms of its ongoing behavior, not by a relation between its initial and final states. Therefore, temporal logic [31], a logic designed to model and reason about time-dependent properties of certain problems, is very suitable for the specification of reactive systems.

For reactive systems, it is also essential to study and analyse each process based on its "local time" or its local clock [13], because processes may be running on clocks with varying rates of progress (especially when they are executed on independent processors). Moreover, for describing such systems naturally, it is essential to introduce the notion of granularity of time into temporal logic, because local clocks associated with different processes in a given system may be different.

TLC [23] is an extension of temporal logic that can be used in applications involving granularity of time or multiple clocks. In this logic, predicates are allowed to be defined on local clocks which are subsequences of the global clock. The notion of a clock naturally extends to formulas of the logic through the use of an appropriate clock calculus. TLC has been used for specifying those systems such as distributed systems [24] and knowledge-based simulation systems [26]. TLC formulas can naturally be used to describe timing properties of reactive systems. In [27], Liu and Orgun presented a framework based on TLC for the formal specification of reactive systems. The formal specification can provide a clear description of the behavior of a reactive system, and its timing properties can be derived from it. Such representations of reactive systems and their timing properties make the verification of those kind of systems more convenient as well as simpler.

In our framework, the behavior of a reactive system is described through a formal specification; and its timing properties, including the safety and liveness properties, are expressed as TLC formulas in a natural way. We represent both local and global timing properties through deducibility relations between the formal specification of the system and formulas representing the properties. In particular, the bounded-invariance and bounded-response properties can also be expressed as TLC formulas. The proposed proof techniques for the verification of timing properties includes an application of local reasoning and deriving fixed-time rules from the proof system of TLC. Local reasoning can be used when a timing property involves a set of events, all of which are defined on the same local clock. Fixed-time rules are very useful in reasoning about single events such as communication between two processes. Under the proposed framework, the
timing properties of a reactive system can also be directly reasoned about by induction from the formal specification of the system.

The rest of this paper is organised as follows. Section 2 discusses related work. Section 3 introduces the temporal logic TLC. We in particular discuss local clocks and outline the semantics of temporal operators and their axiomatization. We also present basic fixed-time rules, which are directly derived from the proof system. Section 4 discusses the formal specification of reactive systems. The specification of a simple send-receive system is also presented to illustrate our method. Section 5 shows how to use TLC formulas to describe timing properties of a system, such as safety and liveness properties. Section 6 presents some important timing properties of the send-receive system. Section 7 discusses verification by local reasoning. In Section 8, we demonstrate the verification of global properties of reactive systems using fixed-time rules. The last section concludes the paper with a brief discussion.

2. Related work

TLC is an extension of temporal logic in which each formula is associated with a local clock, that is, a subsequence of the global timeline modeled by the sequence of natural numbers. TLC offers two temporal operators, first and next [27]. Their intuitive meanings are as follows:

- first $A$: $A$ is true at the initial moment in time,
- next $A$: $A$ is true at the next moment in time.

Note that the meanings of first and next are relative to the clocks of given formulas, not to the global timeline. Temporal modalities $\Box$ (always) and $\Diamond$ (sometime) are also introduced, whose meanings also depend on a given local clock. More details on TLC are given in the next section.

There are also many other temporal logics proposed for specifying and verifying reactive systems. The most popular ones are the logic of Manna and Pnueli [31,33] and TLA (the temporal logic of actions), proposed by Lamport [21]. In the rest of this section, we discuss these temporal logics and some other related work such as branching time logics and partial order logics in more detail.

In [33], transition systems [9] are used as a computation model for reactive systems and temporal logic is used as a specification language to express properties that should be satisfied by any proposed implementation. In [16], two approaches are presented for the specification of timing properties of reactive systems. The first approach, called bounded operator method, introduces one or more time-bounded versions for each temporal operator, such as sometime $\Diamond$ and always $\Box$. For example, the formula $\Diamond_{\leq 5} p$ asserts that $p$ will occur sometime within 5 time units from now. This approach is also used in [2,11,17]. The second approach, called explicit-clock method, uses a time variable as the current time at each state. For example, the requirement of a timed response of $q$ to $p$ within at most 5 time units, can be expressed by the formula $(p \land t = T) \rightarrow \Diamond (q \land t \leq T + 5)$, where the free variable $T$ is used to refer to the current moment in time. The use of the second approach to express timing properties can
also be found in [40]. Manna and Pnueli [33] also propose two types of verification techniques: a deductive approach, which is based on a set of rules that reduce the task of proving a timing property to checking the validity of first-order formulas, and an algorithmic approach, which presents algorithms for automatic verification of timing properties.

TLA [21] is suitable for specifying and reasoning about concurrent algorithms. In the framework based on TLA, systems and their properties are represented as TLA formulas, so the assertion that a system meets its specification and the assertion that one system implements another are both expressed by logic implication. Based on TLA, Abadi and Lamport [1] suggested that the specification of a system is the conjunction of its components' specifications, so that the properties of the system can be verified by reasoning about its components. An approach to specifying concurrent systems based on modules can also be found in [19].

Branching time logics, such as UB [4], CTL [7], CTL* [10] and BCTL [25] can also be used to specify properties of reactive systems. Usually, in a branching time logic, a safety property is expressed as an invariance assertion of the form $\forall \Box A$, which means that $A$ is true at all states of a computation tree [7] or, equivalently, $A$ is always true at all branches of a given branching clock [25]. A liveness property is expressible as an inevitability assertion $\forall \Diamond A$, which says that $A$ is true at some moment in time of all branches on the clock associated with the formula $A$.

Another class of temporal logics which have been used for specifying timing properties of reactive systems are partial order logics, such as ESL-event structure logic [38] and its extensions DESL [39] and ESL[C] [36]. These logics can be used to deal with structures of local states, which allow one to distinguish concurrency of a system from non-determinism. In ESL, the behavior of a concurrent system is represented by an event structure.

In the representation of systems and their properties, our framework is closely related to that based on TLA [21]. However, in our framework, the behavior of a system is represented through a formal specification. Such representations are suitable to express a system or its proposed implementations, which can be given in a range of description languages. There is no need to introduce time-bounded temporal operators or explicit-clocks for the specification of timing properties. For instance, in the above timed response of $p$ to $q$ when $p$ and $q$ have the same local clock, the requirement can be represented by the formula

$$p \rightarrow \bigvee_{n=0}^{5} \text{next}[i] q,$$

where $\text{next}[n]$ denotes $n$ applications of $\text{next}$. Note that, when $n = 0$, $\text{next}[n] q$ denotes $q$. The representation only involves logical symbols of TLC. Also in local reasoning, the proof procedure for proving a timing property only involves a local clock and all formulas involved in the proof are actually defined on the same local clock. The representation of formulas without any time-bounded operators or explicit-clocks therefore simplifies local reasoning.
In our framework based on TLC, given a reactive system $S$ its timing properties are usually represented as deducibility relations of the following form:

$$S \models \phi, A \vdash B,$$

where $S \models \phi$ is the formal specification of the system $S$. This relation states that, if $S \models \phi$ is the formal description of the behavior of the system $S$, or in more general, the description of any proposed implementation of $S$, and $A$ is satisfied under the description, then $B$ is satisfied. Here the local clocks associated with formulas $A$ and $B$ may be the same or different. The actions of a process (or a system) involved may also overlap with the other process (system) actions. Thus, the representation gives us a flexible means for specifying timing properties of systems even when they are based on different local clocks. The task of verifying a timing property is then reduced to checking the validity of the deducibility relation representing the property.

In order to verify timing properties of reactive systems, apart from the use of local reasoning, the use of fixed-time rules is also important; fixed-time rules can be used to deal with the communicating processes of a reactive system. For timing analysis of asynchronous processes of a system, Moller and Tofts proposed a temporal calculus of communicating systems (TCCS) \[35\], in which time is allowed to pass independent of the functional aspects of a process. The language TCCS is a timed extension of CCS, Milner's Calculus of Communicating Systems in \[34\]. In our framework, the formulas representing the behavior of processes of a system are associated with their own local clocks. In reasoning, we can also use fixed-time rules to derive a deducibility relation among formulas with different clocks.

3. Temporal logic TLC

TLC is a linear temporal logic with multiple granularity of time (or multiple clocks). In this logic, each predicate symbol is assigned a local clock, and all formulas can be clocked in terms of the clocks of predicate symbols appearing in them through a clock calculus. In this paper, to make TLC more expressive, we extend it with two basic modalities $\Box$ (always) and $\diamond$ (sometime). We now briefly introduce the logic, including its syntax, semantics and axioms and inference rules. For more details, we refer the reader to the literature \[22,23\].

3.1. Syntax

In the vocabulary of TLC, apart from variables, function symbols and predicate symbols, we have the primitive propositional connectives, $\neg$ and $\land$, universal quantifier, $\forall$, and three temporal operators: first (the initial moment in time), next (the next moment in time), and $\Box$ (always).

In TLC, the definition of terms is as usual \[28\]. We now define temporal atomic formulas, then give the definition of (well-formed) formulas.
Definition 1. Temporal atomic formulas (or simply, atoms) are defined inductively as follows:
- If \( p \) is an \( n \)-ary predicate symbol and \( e_1, \ldots, e_n \) are terms, then \( p(e_1, \ldots, e_n) \) is a temporal atomic formula. Such formulas are in particular called pure temporal atomic formulas or pure atoms.
- If \( A \) is a temporal atomic formula, so are first \( A \) and next \( A \).

Definition 2. A (well-formed) formula is defined inductively as follows:
- All atoms are formulas.
- If \( A \) is a formula, so are \( \neg A \), first \( A \), next \( A \) and \( \Box A \).
- If \( A \) and \( B \) are formulas, so is \( (A \land B) \).
- If \( A \) is a formula and \( X \) is a variable, then \( \forall X A \) is a formula.

The connectives \( \lor, \rightarrow, \leftrightarrow \), and the quantifier \( \exists \) can be derived from the primitive connectives and quantifier as usual. We assume the usual definition of the temporal operator \( \lozenge \) (sometimes) as follows:
\[
\lozenge A \equiv \neg \Box \neg A
\]

In addition, we may use the notation next\([n]\) to denote \( n \) applications of next's, where \( n \) is a natural number. For instance, first next next \( A \) and first next [2] \( A \) actually represent the same formula. If \( n = 0 \), then next\([n]\) is the empty string. We also write
\[
\bigwedge_{i=0}^{n} \text{next}[i] A \quad \text{and} \quad \bigvee_{i=0}^{n} \text{next}[i] A
\]
as the abbreviations of the formulas \( A \land \text{next} A \land \text{next}[2] A \land \cdots \land \text{next}[n] A \) and \( A \lor \text{next} A \lor \text{next}[2] A \lor \cdots \lor \text{next}[n] A \), respectively.

3.2. Clocks

Let \( \omega \) denote the set of natural numbers \( \{0, 1, 2, 3, \ldots\} \). In TLC, clocks are defined as sequences over \( \omega \). Formally, we have the following definition:

Definition 3. The global clock is the increasing sequence of natural numbers, i.e., \( \langle 0, 1, 2, \ldots \rangle \). A local clock is a subsequence of the global clock, that is, a strictly increasing sequence of natural numbers, either finite \( \langle t_0, t_1, t_2, \ldots, t_n \rangle \) or infinite: \( \langle t_0, t_1, t_2, \ldots \rangle \). In particular, the global clock and the empty clock, denoted by gck and \( \langle \rangle \) respectively, are also local clocks.

Let \( t \in \text{ck} \) denote the fact that \( t \) is a moment in time on the clock \( \text{ck} \). Let \( \mathcal{C} \) be the set of all clocks and \( \subseteq \) be an ordering relation on the elements of \( \mathcal{C} \), defined as follows: for any \( \text{ck}_1, \text{ck}_2 \in \mathcal{C} \), \( \text{ck}_1 \subseteq \text{ck}_2 \) if and only if for all \( t \in \text{ck}_1 \) we have that \( t \in \text{ck}_2 \). It is easy to show that \( (\mathcal{C}, \subseteq) \) is a complete lattice. Therefore, we can define two operations that are analogous to set intersection and union:
\[
\text{ck}_1 \cap \text{ck}_2 \overset{\text{def}}{=} \text{g.l.b.}\{\text{ck}_1, \text{ck}_2\},
\]
\[ ck_1 \sqcup ck_2 \overset{\text{def}}{=} \text{l.u.b.}\{ck_1, ck_2\}, \]

where \( ck_1, ck_2 \in \mathcal{X} \), \( \text{g.l.b.} \) stands for "the greatest lower bound" and \( \text{l.u.b.} \) for "the least upper bound" under the relation \( \sqsubseteq \). By definition, \( ck_1 \sqcap ck_2 \) is a local clock consisting of those moments in time which appear on both \( ck_1 \) and \( ck_2 \), and \( ck_1 \sqcup ck_2 \) is a local clock consisting of those moments in time which appear on either \( ck_1 \) or \( ck_2 \) or both.

We use a clock assignment, a map from the set of predicate symbols to the set of clocks, to assign a local clock to each predicate symbol.

**Definition 4.** A clock assignment \( ck \) of TLC is a map from the set \( \mathcal{P} \) of predicate symbols to the set \( \mathcal{X} \) of clocks, i.e. \( ck \in [\mathcal{P} \rightarrow \mathcal{X}] \). The notation \( ck(p) \) denotes the clock which is associated with a predicate symbol \( p \) on a given clock assignment \( ck \).

For any formula \( A \), its local clock over a given clock assignment is defined based on its syntactic structure and the clocks of predicate symbols appearing in it. We now extend the notion of a clock assignment to formulas of TLC. Note that the clock of a given formula does not contain any moments that are on the clock of at least one of the predicate symbols appearing in it; in other words, no new moments are created.

**Definition 5 (Clock calculus).** Let \( A \) be a formula and \( ck \) a clock assignment. The local clock associated with \( A \), denoted as \( ck_A \), is defined inductively as follows:

- If \( A \) is a pure atom \( p(x_1, \ldots, x_n) \), then \( ck_A = ck(p) \).
- If \( A = \text{first} B, \text{not} B, \text{\&} B \) or \( (\forall x)B \), then \( ck_A = ck_B \).
- If \( A = (B \lor C) \), then \( ck_A = ck_B \sqcap ck_C \).
- If \( A = \text{next} B \), then (1) \( ck_A = (t_0, t_1, \ldots, t_{n-1}) \) when \( ck_B = (t_0, t_1, \ldots, t_n) \) is non-empty and finite; (2) \( ck_A = ck_B \) when \( ck_B \) is infinite or empty.

Note that the difference between \( ck_{\text{next} A} \) and \( ck_A \) is only that \( ck_A \) has one more element than \( ck_{\text{next} A} \) does when \( ck_A \) is finite. Actually, in this case, \( ck_{\text{next} A} \) is generated by deleting the last element in \( ck_A \) because the last element \( t_n \) does not have a next moment defined for it.

In the following, we use the notation \( ck_0, ck_1, \ldots, \) or \( ck_A, ck_B, \ldots \) to represent local clocks. To simplify the presentation, we also use the notation \( ck_B \sqcap ck_C \) to represent \( ck_{B, C} \) and \( ck_{\text{next} A} \) to represent \( ck_A \).

By the definitions, it is easy to show that, for the derived connectives, we have that \( ck_{B \lor C} = ck_{B \rightarrow C} = ck_{B \lor C} = ck_{B, C} \); for the existential quantifier \( \exists \), we have \( ck_{(\exists x)B} = ck_B \), and for \( \forall \), we have \( ck_{\forall B} = ck_B \).

**Lemma 1.** Let \( A \) be a formula and \( ck \) a clock assignment. Then \( ck_A \in \mathcal{X} \) (i.e., every formula of TLC can be clocked).

**Proof.** By induction on the structure of formulas. ⊓
Definition 6. Given a local clock \( ck_i = (t_0, t_1, t_2, \ldots) \), we define the rank of \( t_n \) on \( ck_i \) to be \( n \), written as \( \text{rank}(t_n, ck_i) = n \). Inversely, we write \( t_n = ck_i^{(n)} \), which means that \( t_n \) is the moment in time on \( ck_i \) whose rank is \( n \).

For the global clock denoted as \( gck \), we have that \( \text{rank}(t, gck) = t \) and \( gck^{(t)} = t \).

The following lemma will help to understand the semantics for TLC, given in the next section.

Lemma 2. Let \( A \) be a formula. Then, for any \( t \in ck_A \), we have that \( t \in ck_A \) and \( \text{rank}(t, ck_A) = \text{rank}(t, ck_A) \).

Proof. Straightforward from Definitions 5 and 6. \( \Box \)

3.3. Semantics

In TLC, the meaning of a predicate symbol \( p \) is actually a partial mapping from \( \omega \) to \( P(D^n) \) where \( n \) is the arity of \( p \), \( D \) is the domain of discourse, \( D^n \) is the \( n \)-folded Cartesian product of \( D \), and \( P(D^n) \) is the power set of \( D^n \). Under any given clock assignment, for any \( t \in ck(p) \), the mapping is naturally defined, i.e., there will be a corresponding subset of \( D^n \); otherwise, the image is undefined. A formula is defined only over the moments in time appearing on its clock over a given clock assignment. Therefore, for a given time \( t \in \omega \), the value of the formula can be true, false, or undefined, depending on the local clock associated with it by Definition 5.

A temporal interpretation together with a clock assignment assigns meanings to all the basic elements of TLC. In the definition, the notation \([X \rightarrow Y]\) denotes the set of functions from set \( X \) to set \( Y \).

Definition 7. A temporal interpretation \( I \) on a given clock assignment \( ck \) of TLC comprises a non-empty set \( D \), called the domain of the interpretation, over which the variables range, together with for each variable, an element of \( D \); for each \( n \)-ary function symbol, an element of \([D^n \rightarrow D]\); and for each \( n \)-ary predicate symbol \( p \), an element of \([ck(p) \rightarrow P(D^n)]\).

Now we give the definition of the satisfaction relation \( \models \). Below, the notation \( \models_{I, ck, t} A \) denotes the fact that a formula \( A \) is true at \( t(\in ck_i) \) under a temporal interpretation \( I \).

Let \( p \) be an \( n \)-ary predicate and \( D \) be the domain of an interpretation \( I \). We use \( I(p)(t) \) to denote the element of \( P(D^n) \) that is assigned to \( p \) at \( t(\in ck(p)) \) under the interpretation \( I \).

Definition 8. Let \( I \) be a temporal interpretation on a given clock assignment \( ck \) of TLC, and \( A \) and \( B \) formulas of TLC. The semantics of elements of TLC are given inductively as follows:

(1) If \( f(e_1, \ldots, e_n) \) is a term where \( f \) is an \( n \)-ary function symbol, then \( I(f(e_1, \ldots, e_n)) = I(f)(I(e_1), \ldots, I(e_n)) \).
(2) For any $n$-ary predicate symbol $p$ and terms $e_1, \ldots, e_n$ and any $t \in ck(p)$, $\models_{I, ck_i, t} p(e_1, \ldots, e_n)$ if and only if $(I(e_1), \ldots, I(e_n)) \in I(p)(t)$.

(3) For any $t \in ck_A$, $\models_{I, ck_i, t} \neg A$ if and only if it is not the case that $\models_{I, ck_i, t} A$.

(4) For any $t \in ck_A, B$, $\models_{I, ck_i, t} (A \land B)$ if and only if $\models_{I, ck_i, t} A$ and $\models_{I, ck_i, t} B$.

(5) For any $t \in ck_A$, $\models_{I, ck_i, t} (\forall x)A$ if and only if $\models_{[d/x], ck_i, t} A$ for all $d \in D$ where the interpretation $I[d/x]$ is just like $I$ except that the variable $x$ is assigned the value $d$ in $I[d/x]$.

(6) For any $t \in ck_A$, $\models_{I, ck_i, t} \text{first } A$ if and only if $\models_{I, ck_i, t_0} A$, where $t_0 = ck_A^{(0)}$.

(7) For any $t \in ck_A$, $\models_{I, ck_i, t} \text{next } A$, i.e., $\models_{I, ck_i, t} \text{next } A$ if and only if $\models_{I, ck_i, t_{i+1}} A$, where $i = \text{rank}(t, ck_A)$.

Note that, because $ck_A, B = ck_A \cap ck_B$, if $t \in ck_A, B$ we must have that $t \in ck_A$ and $t \in ck_B$. We can therefore define (4) as above. Also, for any $t \in ck_{\bigcap A}$, by lemma 2, we have that $t \in ck_A$. When $t$ refers to the current moment in time on the clock $ck_A$, its next moment on the same clock must exist. Therefore, we can define (7) as above.

When $ck_i$ is not empty, we use the notation $\models_{I, ck_i} A$ to denote the fact that $A$ is true on a local clock $ck_i$ under $I$, in other words, $\models_{I, ck_i} A$ if and only if $\models_{I, ck_i, t} A$ for all $t \in ck_i$. We also use the notation $\models_{ck_i} A$ to denote the fact that $\models_{I, ck_i} A$ for any temporal interpretation $I$. In particular, if $\models_{I, ck_i} A$, then we say that the temporal interpretation $I$ on $ck$ is a model of the formula $A$. We use the notation $\models_I A$ to denote the fact that $I$ is a model of the formula $A$, and also use $\models A$ to denote the fact that for any model $I$ we have $\models_I A$.

The semantics of $\lor$, $\rightarrow$, $\leftrightarrow$ and $\exists$ can be derived based on those of the primitive connectives. We must also define the semantics of temporal operator $\Box$, otherwise the definition of $\models$ is not complete. Intuitively, $\Box A$ is true at a moment $t$ on the local clock $ck_A$ just in the case $A$ is true at all moments in time on $ck_A$. Formally, we have that:

(8) For any $t \in ck_A$, $\models_{I, ck_i, t} \Box A$ if and only if $\models_{I, ck_i} A$.

We could also introduce the temporal operator $\Diamond$ directly into TLC. Then, the semantics of $\Diamond$ could be easily obtained from definitions (3) and (8).

(9) For any $t \in ck_A$, $\models_{I, ck_i, t} \Diamond A$ if and only if $\models_{I, ck_i, s} A$ for some $s \in ck_A$.

Intuitively, this indicates that $\Diamond A$ is true at a moment $t$ on $ck_A$ just in the case $A$ is true at some moment in time on $ck_A$.

In TLC, the meanings of $\Box A$ and $\Diamond A$ are not relative to particular moments (current time). In other words, from the view of an investigator, if $A$ is true everywhere on the local clock $ck_A$, then $\Box A$ is true at any moment on $ck_A$. Similarly, if $A$ is true at some moment on the clock $ck_A$, then $\Diamond A$ is true at any moment on $ck_A$. We can also show that the following lemma holds.

**Lemma 3.** Let $A$ be a formula. Then we have the following:

1. $\models \Box A$ if and only if $\models \text{first } A$ and $\models \Box (\text{next } A)$.
2. $\models \Diamond A$ if and only if $\models \text{first } A$ or $\models \Diamond (\text{next } A)$.
Proof. By definition, we have that \( \models \Box A \)
if and only if \( \Box A \) is true at any moment in time on \( ck_A \)
if and only if \( A \) is true at all moments in time on \( ck_A \)
if and only if \( A \) is true at \( ck_A^{(0)} \) and \( A \) is true at all moments in time other than \( ck_A^{(0)} \)
if and only if \( \text{first} A \) is true at any moments in time on \( ck_A \) and \( \text{next} A \)
is true at all moments in time on \( ck_{\Diamond A} \)
if and only if \( \text{first} A \) and \( \Box (\text{next} A) \).

Thus, the proof of (1) is completed. The proof of (2) can be obtained in a similar fashion. \( \Box \)

This lemma suggests that both \( \Box \) and \( \Diamond \) could be recursively defined as follows:

\[ \Box A \overset{\text{def}}{=} \text{first} A \land \Box (\text{next} A) \]

\[ \Diamond A \overset{\text{def}}{=} \text{first} A \lor \Box (\text{next} A) \]

However, it is not hard to see that such definitions may lead to infinitary formulas.

We now consider a few examples of properties of programs expressible in TLC. We assume that, in the two kinds of properties given below, all predicates which occur in the same formula are associated with the same local clock.

1. Invariance (safety) properties:

\[ \Box A \] all states reached by a program satisfies \( A \)
\[ \Box \text{next}[n] A \] all states reached by a program satisfies \( A \) from the moment \( ck_A^{(n)} \)
\[ \Box (\neg A) \lor (\neg B) \] a program cannot enter critical regions \( A \) and \( B \) simultaneously
\[ A \rightarrow \Box B \] if \( A \) is satisfied at the current moment in time, all states reached by a program satisfies \( B \)
\[ A \rightarrow \bigwedge_{i=0}^{n} \text{next}[i] B \] no such state satisfying \( A \) is followed by a state satisfying \( \neg B \) until the \( n \)th next moment from now

2. Eventuality (liveness) properties:

\[ \Diamond A \] there is at least one state reached by a program that satisfies \( A \)
\[ \Diamond \text{next}[n] A \] there is at least one state reached by a program that satisfies \( A \) from the moment \( ck_A^{(n)} \)
\[ A \rightarrow \Diamond B \] there is a state \( B \) reached by a program after the state satisfying \( A \)
\[ A \rightarrow \bigvee_{i=0}^{n} \text{next}[i] B \] every state satisfying \( A \) is followed by a state satisfying \( B \) before the \( n \)th next moment from now

The last invariance property is a bounded-invariance property; and the last eventualitity property is a bounded-response property [16]. The properties involved in several different local clocks are discussed in the following sections.

Some other examples of timing properties expressible by means of temporal logic can also be found in [16, 18, 29, 31, 33]. A classification of program properties, including the past time operators, can be found in [30, 33].
3.4. Proof system for TLC

The proof system for TLC consists of a set of axioms and a set of inference rules. Apart from the axioms of first-order logic and substitution (universal instantiation), TLC also has the following axioms and inference rules, which are related to the temporal operators and clock assignment. Below, the notation \( \vdash A \) denotes the fact that \( A \) is a theorem of TLC. All theorems of the form \( \vdash A \) hold on the local clock associated with the formula \( A \), i.e., \( ck_A \), under any given clock assignment \( ck \). They do not necessarily hold on an arbitrary clock. In other words, \( \vdash A \) means \( \vdash_{ck_A} A \), i.e., \( A \) is a theorem that holds on the local clock \( ck_A \) for any clock assignment \( ck \).

**Axioms**

A1. \( \vdash \) first first \( A \) \( \leftrightarrow \) first \( A \).

A2. \( \vdash \) next first \( A \) \( \leftrightarrow \) first \( A \), when \( ck_A \) is infinite.

A3. \( \vdash \) first \( (\neg A) \) \( \leftrightarrow \) \( \neg \) (first \( A \)).

A4. \( \vdash \) next \( (\neg A) \) \( \leftrightarrow \) \( \neg \) (next \( A \)).

A5. \( \vdash \) first \( (\forall x) (A) \) \( \leftrightarrow \) (\( \forall x \)) (first \( A \)).

A6. \( \vdash \) next \( (\forall x) (A) \) \( \leftrightarrow \) (\( \forall x \)) (next \( A \)).

A7. \( \vdash \) first \( (A \land B) \) \( \leftrightarrow \) (first \( A \)) \( \land \) (first \( B \)), when \( ck_A^{(0)} = ck_B^{(0)} \).

A8. \( \vdash \) next \( (A \land B) \) \( \leftrightarrow \) (next \( A \)) \( \land \) (next \( B \)), when \( ck_A = ck_B \).

A9. \( \vdash \) □ A \( \leftrightarrow \) □ A.

A10. \( \vdash \) first □ A \( \leftrightarrow \) □ A.

A11. \( \vdash \) next □ A \( \leftrightarrow \) □ A, when \( ck_A \) is infinite.

A12. \( \vdash \) □(A \( \land \) B) \( \leftrightarrow \) (□ A \( \land \) □ B), when \( ck_A = ck_B \).

**Inference rules**

R1. If \( \vdash A \rightarrow B \) and \( \vdash A \), then \( \vdash B \), when \( ck_A = ck_B \).

R2. If \( \vdash A \), then \( \vdash \) first \( A \), when \( ck_A \) is non-empty.

R3. If \( \vdash A \), then \( \vdash \) next \( A \), when \( ck_A \) is non-empty.

R4. If \( \vdash A \), then \( \vdash \) □ A.

Rule R1 can be viewed as Modus Ponens (MP); Rules R2–R4 are called temporal operator introduction rules.

The correctness (soundness) of the axioms and the rules is straightforward. Therefore we state the following result without proof.

**Lemma 4** (Liu [22]). The axioms A1–A12 and the rules R1–R4 are valid with respect to the semantics scheme for TLC.

3.5. Fixed-time rules

Fixed-time rules are derived from the proof rules of TLC and they very useful in reasoning about single events such as communication between two processes. We say that an atom is fixed-time if it has an application of first followed by a number of applications of nexts. Any fixed-time atom is fixed to some moment in time on its local clock.
Definition 9. Let \( p(e_1, \ldots, e_k) \) be a pure atom, \( ck \) a clock assignment, and \( ck_p \) the clock associated with \( p \) over \( ck \). If \( t < ck_p \) and \( \text{rank}(t, ck) = n \) where \( n \) is a natural number, then we call \( \text{first next}[n] \) \( p(e_1, \ldots, e_k) \) a fixed-time atom and \( ck_p^{(n)} \) (i.e. \( t \)) the current time of the atom \( p(e_1, \ldots, e_k) \) with respect to the fixed-time atom.

For example, suppose that \( p(x) \) is an atom, and \( ck(p) = (2, 5, 8, \ldots) \). Then, \( \text{first} \ p(x) \) is fixed to moment 2, \( \text{first next} \ p(x) \) is fixed to moment 5 and so on.

We also have the notion of fixed-time formulas:

Definition 10. Let \( A \) be a formula, \( ck \) a clock assignment and \( n \in \omega \). Suppose that \( ck_A^{(n)} \) exists. We call \( \text{first next}[n] \) \( A \) a fixed-time formula, which is fixed to time \( ck_A^{(n)} \), and also say that \( ck_A^{(n)} \) is the current time of the formula \( A \) with respect to the fixed-time formula.

Lemma 5. Let \( A \) be a formula, \( ck \) a clock assignment and \( n \in \omega \). If \( ck_A^{(n)} \) exists, then we have that

\[
\models_{ck} \text{first next}[n] A \text{ if and only if } A \text{ is true at } ck_A^{(n)}.
\]

Proof. We first prove necessity. Assume that \( \models_{ck} \text{first next}[n] A \), then, by Definition 8, we have that \( \text{next}[n] A \) is true at \( ck_A^{(n)} \). Therefore, all the following assertions hold:

\[
\begin{align*}
\text{next}[n - 1] A & \text{ is true at } ck_A^{(n - 1)}, \\
\text{next}[n - 2] A & \text{ is true at } ck_A^{(n - 2)}, \\
& \ldots \\
\text{next}[2] A & \text{ is true at } ck_A^{(n - 2)}, \\
\text{next} A & \text{ is true at } ck_A^{(n - 1)}, \\
A & \text{ is true at } ck_A^{(n)}.
\end{align*}
\]

The last assertion is what we need to show.

Inversely, if \( A \) is true at \( ck_A^{(n)} \), then we have that

\[
\begin{align*}
\text{next} A & \text{ is true at } ck_A^{(n - 1)}, \\
\text{next}[2] A & \text{ is true at } ck_A^{(n - 2)}, \\
& \ldots \\
\text{next}[n] A & \text{ is true at } ck_A^{(0)}.
\end{align*}
\]

The last assertion implies that \( \text{first next}[n] A \) is true at any moment in time on the clock \( ck_F \), where \( F = \text{first next}[n] A \). Therefore we have that \( \models_{ck} \text{first next}[n] A \). □

Let \( A \) and \( B \) be formulas and \( ck \) a clock assignment. If \( ck_A = ck_B \), then, according to axioms A7 and A8, it is not difficult to show that, for any \( n \in \omega \),

\[
\models_{ck} \text{first next}[n] (A \land B) \text{ if and only if } \\
\models_{ck} \text{first next}[n] A \text{ and } \models_{ck} \text{first next}[n] B
\]

when there exists \( t \in ck_{A,B} \) such that \( n = \text{rank}(t, ck_{A,B}) \).
If $c_k A \neq c_k B$, the above assertion does not hold. However, we have the following result:

**Lemma 6.** Let $A$ and $B$ be formulas, $c_k$ a clock assignment $t \in c_k A, B$. Assume that $n = \text{rank}(t, c_k A, B)$, $m = \text{rank}(t, c_k A)$ and $k = \text{rank}(t, c_k B)$, then

$$
\models c_k \text{ first next}[n] (A \land B) \text{ if and only if } \\
\models c_k \text{ first next}[m] A \text{ and } \models c_k \text{ first next}[k] B.
$$

**Proof.** Without loss of generality, we assume that

$$
c_k A, B = (t_0, t_1, \ldots, t_n, \ldots),
$$

$$
c_k A = (s_0, s_1, \ldots, s_m, \ldots),
$$

$$
c_k B = (h_0, h_1, \ldots, h_k, \ldots),
$$

where $t_n = s_m = h_k = t$. By Lemma 5, we have that

$$
\models c_k \text{ first next}[n] (A \land B) \\
\text{ if and only if } (A \land B) \text{ is true at } t_n, \text{ i.e., } \\
\text{ if and only if } A \text{ is true at } t \text{ and } B \text{ is true at } t \\
\text{ if and only if } A \text{ is true at } s_m \text{ and } B \text{ is true at } h_k \\
\text{ if and only if } \models c_k \text{ first next}[m] A \text{ and } \models c_k \text{ first next}[k] B.
$$

Thus, we complete the proof of the lemma. □

This lemma can be used to deal with such cases when there are some formulas associated with different local clocks. Based on the lemma, we can directly obtain the following rules of inference. Here, a rule with the form $\Gamma \vdash A$, where $A$ is a formula and $\Gamma$ a set of formulas. For any $t \in c_k A, B$, we have the following rules:

1. (F →₁) $\text{first next}[k] (B \rightarrow A)$, where $k = \text{rank}(t, c_k A, B)$
   
   $\text{first next}[m] B$, where $m = \text{rank}(t, c_k B)$
   
   $\text{first next}[n] A$, where $n = \text{rank}(t, c_k A)$

2. (F →₂) $\text{first next}[m] B$, where $m = \text{rank}(t, c_k B)$
   
   $\text{first next}[n] A$, where $n = \text{rank}(t, c_k A)$
   
   $\text{first next}[k] (B \rightarrow A)$, where $k = \text{rank}(t, c_k A, B)$

3. (F ∧₁) $\text{first next}[n] A$, where $n = \text{rank}(t, c_k A)$
   
   $\text{first next}[m] B$, where $m = \text{rank}(t, c_k B)$
   
   $\text{first next}[k] (A \land B)$, where $k = \text{rank}(t, c_k A, B)$

4. (F ∧₂) $\text{first next}[k] (A \land B)$, where $k = \text{rank}(t, c_k A, B)$
   
   $\text{first next}[n] A$, where $n = \text{rank}(t, c_k A)$

These rules are only involved in fixed-time formulas; they deal with inference about those formulas that are fixed to a particular time $t$. Therefore, we call them fixed-time rules.
Note that for any rule given above, if we have \( t \in \text{ck}_{A,B} \), then we must have \( t \in \text{ck}_A \) and \( t \in \text{ck}_B \). Therefore, there must exist \( n, m \) and \( k \) such that \( n = \text{rank}(t, \text{ck}_A) \), \( m = \text{rank}(t, \text{ck}_B) \) and \( k = \text{rank}(t, \text{ck}_{A,B}) \). Fixed-time rules can be applied for reasoning involved in several formulas associated with different local clocks.

**Lemma 7.** The fixed-time rules \( F \rightarrow_1 \), \( F \rightarrow_2 \), \( F \wedge_1 \) and \( F \wedge_2 \) are valid with respect to the semantics scheme for TLC.

**Proof.** We now prove the validity of the rule \( F \rightarrow_1 \). For any interpretation \( I \) and any clock assignment \( \text{ck} \), by the semantics scheme for TLC, we have that

\[
\models_I,\text{ck} \text{first } \text{next}[k] (B \rightarrow A) \text{ if and only if } \\
\models_I,\text{ck},t_1 (B \rightarrow A), \text{ where } t_1 = \text{ck}^{k}_{B \rightarrow A}. \\
\models_I,\text{ck} \text{first } \text{next}[m] B \text{ if and only if } \models_I,\text{ck},t_2 B, \text{ where } t_2 = \text{ck}^{m}_{B}.
\]

Thus, if \( \models_I,\text{ck} \rightarrow_4 \text{first } \text{next}[k] (B \rightarrow A) \) and \( \models_I,\text{ck} \text{first } \text{next}[m] B \), and we have that \( t_1 = t_2 = t \), then we have that \( \models_I,\text{ck},t (B \rightarrow A) \) and \( \models_I,\text{ck},t B \).

On the other hand, due to the fact that \( t \in \text{ck}_{B \rightarrow A} \), we must have that \( t \in \text{ck}_A \). Therefore, we have \( \models_I,\text{ck},t A \). Assuming \( n = \text{rank}(t, \text{ck}_A) \), we have that \( \models_I,\text{ck},t \text{first } \text{next}[n] A \). This completes the proof for the rule \( F \rightarrow_1 \).

The validity of the remaining rules \( F \rightarrow_2 \), \( F \wedge_1 \) and \( F \wedge_2 \) can be shown in a similar fashion. \( \Box \)

### 4. Specifying reactive systems

A formal specification of a given reactive system provides a whole description of its behavior as well as its initial conditions in a given implementation environment. Such a formal specification is the basis for verifying timing properties of the system. This section first develops an approach to the formal specification of reactive systems based on TLC, and then presents a simple send-receive system to illustrate the specification. In Sections 6–8, we discuss and prove some important timing properties of the system.

#### 4.1. Formal specification

We assume that a reactive system consists of a collection of processes communication with one another through channels. Each process may be running on its own local clock. The activity of each process is modeled as executing a sequence of events. Formally, we have the following definition of an event:

**Definition 11.** An event is an action and/or a state with the moment in time at which the action and/or the state happens. Events are represented by ground, fixed-time formulas in TLC. In particular, we call an event of the form \( \text{first } \text{next}[n] A \) an atomic event if \( A \) is a pure atom.

Typically, an event may be executing a single statement such as receiving a message from a channel by a process at a given moment in time. For example, we may have
the formula

\[ \text{first next}[2] \text{ execute}_p(l_1) \]

which indicates the event that, at the second next moment after the initial moment in time, the process \( p_1 \) executes the statement \( l_1 \). Note that the timing is based on the local clock of the predicate \( \text{execute}_p \), i.e., the local clock associated with the process.

An event may be either internal to a process and cause only a local state change, or it may involve communication with another process. We assume that communication is performed by sending and receiving messages along channels, each of which connects two processes: one is the sending process, the other the receiving process.

We consider reactive systems based on an asynchronous communication model. After sending a message, the sending process does not wait for the receiving process to receive the message. Since the message exchange is not synchronised, communication requires buffering for the messages that have been sent but not yet received. Communication between processes is performed through channels. We assume that the capacity of buffers, i.e., the capacity of channels is limited. Therefore, if the receiving process does not respond to a message from the buffer in time, the message may be lost. Without loss of generality, we also assume that communication is performed through the event "send a message \( M \)" and "receive the message \( M \)" which match based on the message identifier \( M \). This means that, if several processes send the same data value to the same process, the messages are unique.

**Definition 12.** A formal specification of a reactive system \( S \), denoted as \( S \mathcal{P} \mathcal{E} \), is a 5-tuple \( \langle \mathcal{V}, \mathcal{P}, \mathcal{Ck}, \mathcal{R}, \mathcal{I} \rangle \), which consists of the following components:
- a finite set \( \mathcal{V} \) of variables each of which ranges over a given domain. The variables are intended to represent states or actions.
- a set \( \mathcal{P} \) of predicate symbols which describe the behavior of the system.
- a clock assignment \( \mathcal{Ck} \), which is a map from the set \( \mathcal{P} \) of predicate symbols to the set \( \mathcal{C} \mathcal{K} \) of clocks.
- a set \( \mathcal{R} \) of rules, represented as TLC formulas, which describe how each action changes the state of the system and the control of actions.
- a set \( \mathcal{I} \) of initial conditions, represented as fixed-time atoms, which are true when the system starts running.

Actual systems are usually composed of several independent processors, each of them executing its own processes. Therefore, it is necessary to consider a local clock for each process involved in the system, so we introduce a clock assignment to the specification. The clock assignment assigns local clocks for all predicates that describe the behavior of the system.

### 4.2. An example: the sr-system

Let us consider a simple system, called the \( sr \)-system (the send–receive system), which consists of two parallel processes, \( p_1 \) and \( p_2 \). Here process \( p_1 \) is the sending
process; it repeatedly picks a number then sends this number to \( p_2 \) via a channel, whose maximum capacity is \( d \), say 5 (i.e., it contains at most 5 messages). Process \( p_2 \) is the receiving process; it repeatedly receives a number \( Y \) then processes it. The action “pick” can be viewed as getting a number from an input device without any constraints; the “process” may be any operation we want to do. The system might be represented in a conventional programming language as follows:

\[
\text{cobegin}
\text{loop pick}(X); \text{send}(X) \text{ end loop}
\|\]
\[
\text{loop receive}(Y); \text{process}(Y) \text{ end loop}
\text{coend}
\]

where \( \| \) means parallel composition, that is, the two processes can be executed in parallel.

In the following, we provide a formal specification \( S^{PE}_{sr} = (V, P, ck, \mathcal{R}, I) \) for the \( sr \)-system. We first need the following variables:

- \( X, Y, M \) range over the set of integers
- \( W \) the control variable of process \( p_1 \), which ranges over \{pick, send\}
- \( Z \) the control variable of process \( p_2 \), which ranges over \{receive, process\}
- \( L \) the state variable of the channel named as \( ch \), which represents a list of numbers

Here \( X \) and \( Y \) are the state variables of \( p_1 \) and \( p_2 \), respectively, and \( E \) is intended to represent elements in the channel. The predicates are:

- \( \text{state}_{p_1}(X) \) the state predicate of process \( p_1 \)
- \( \text{state}_{p_2}(Y) \) the state predicate of process \( p_2 \)
- \( \text{action}_{p_1}(W) \) \( p_1 \) is currently executing the action \( W \)
- \( \text{action}_{p_2}(Z) \) \( p_2 \) is currently executing the action \( Z \)
- \( \text{ch}(L) \) the value of channel \( ch \) is the list \( L \)
- \( \text{head}(M,L) \) \( M \) is the first element of the list \( L \)
- \( \text{equal}(L,L_1) \) \( L \) is equal to the list \( L_1 \)
- \( \text{incoming}(L,M) \) the value of channel \( ch \) is changing by incoming data \( M \)
- \( \text{outgoing}(L,M) \) the value of channel \( ch \) is changing by outgoing data \( M \)

The local clock for each process as well as for each predicate is given based on the implementation environment of the system. Apart from the global clock \( gck = \{0, 1, 2, 3, \ldots\} \), we also have the clocks

\[
ck_1 = \{0, 2, 4, 6, \ldots\} \quad \text{and} \quad ck_2 = \{1, 3, 5, 7, \ldots\},
\]

which are assumed as the local clocks of the processes \( p_1 \) and \( p_2 \) respectively. Thus, the clock assignment \( ck \) may be defined as follows:

\[
ck(\text{state}_{p_1}) = ck_1, \quad ck(\text{action}_{p_1}) = ck_1,
ck(\text{state}_{p_2}) = ck_2, \quad ck(\text{action}_{p_2}) = ck_2,
\]
(n: even number; pi: pick; s: send)

(m: odd number; r: receive; pr: process)

(n, m: as above; in: incoming; out: outgoing)

Fig. 1 shows the behavior of the system: process $p_1$ takes actions pick and send alternately at the moments in time on the clock $ck_1$, process $p_2$ takes actions receive and process at the moments in time on $ck_2$, and the channel gets changes depending on the actions send and receive which are executed by $p_1$ and $p_2$, respectively, at the corresponding moments in time on the global clock.

The behavior of the $sr$-system can formally be described by a set of rules, which is the main component of the specification. All the rules of the system are listed in Fig. 2. In the figure, the symbol $\lambda$ denotes the empty list. Note that we do not list the rules for list operations $*$ and $/$. We assume that such rules are all built-in. Here we use $L*M$ to denote the list obtained by attaching $M$ to the end of the list $L$ and $M/L$ to denote the list obtained by deleting the head $M$ from the list $L$. We may also use $L*L'$ to denote the list obtained by appending $L'$ to the end of the list $L$.

Rules (r1)–(r3) tell us that the actions pick and send are alternately executed by process $p_1$ and there are no cases in which both actions are executed at the same moment in time. Note that we actually assume that the action pick is always successfully executed without any waiting. The rule (r4) says that if $p_1$ is currently executing the actions pick, then the value of its state variable at the next moment in time is $\chi$ which is the number picked. Similarly, from rules (r5)–(r8), we know that the actions receive and process may be alternately executed by the process $p_2$ and there is no case in which the two actions are executed at the same moment in time, but if the list in the channel $ch$ is empty at the current moment in time and the action which is executed is receive, then the process $p_2$ will try the action receive again at the next moment. The rule (r9) says that sending a message $M$ and changing the value of the channel by the
(r1) \( \text{action}_{p1}(\text{pick}) \rightarrow \text{next action}_{p1}(\text{send}) \).

(r2) \( \text{action}_{p1}(\text{send}) \rightarrow \text{next action}_{p1}(\text{pick}) \).

(r3) \( \neg \text{action}_{p1}(\text{send}) \lor \neg \text{action}_{p1}(\text{pick}) \).

(r4) \( (\exists \mathcal{X}) (\text{action}_{p1}(\text{pick}) \rightarrow \text{next state}_{p1}(\mathcal{X})) \).

(r5) \( (\forall \mathcal{L})(\text{action}_{p2}(\text{receive}) \land \text{ch}(\mathcal{L}) \land \text{equal}(\mathcal{L}, \mathcal{A}) \rightarrow 
\text{next action}_{p2}(\text{receive})) \).

(r6) \( (\forall \mathcal{L})(\text{action}_{p2}(\text{receive}) \land \text{ch}(\mathcal{L}) \land \neg \text{equal}(\mathcal{L}, \mathcal{A}) \rightarrow 
\text{next action}_{p2}(\text{process})) \).

(r7) \( \neg \text{action}_{p2}(\text{process}) \rightarrow \text{next action}_{p2}(\text{recvive}) \).

(r8) \( \neg \text{action}_{p2}(\text{receive}) \lor \neg \text{action}_{p2}(\text{process}) \).

(r9) \( (\forall \mathcal{L})(\forall \mathcal{M})(\text{state}_{p1}(\mathcal{M}) \land \text{ch}(\mathcal{L}) \land \text{action}_{p1}(\text{send}) \rightarrow \text{incoming}(\mathcal{L}, \mathcal{M})) \).

(r10) \( (\forall \mathcal{L})(\forall \mathcal{M})(\text{ch}(\mathcal{L}) \land \text{head}(\mathcal{M}, \mathcal{L}) \land \text{action}_{p2}(\text{receive}) \rightarrow 
\text{outgoing}(\mathcal{L}, \mathcal{M})) \).

(r11) \( (\forall \mathcal{L})(\forall \mathcal{M})(\text{ch}(\mathcal{L}) \land \text{head}(\mathcal{M}, \mathcal{L}) \land \text{action}_{p2}(\text{receive}) \rightarrow 
\text{next state}_{p2}(\mathcal{M})) \).

(r12) \( (\forall \mathcal{L})(\forall \mathcal{M})(\text{incoming}(\mathcal{L}, \mathcal{M}) \rightarrow \text{next ch}(\mathcal{L}, \mathcal{M})) \).

(r13) \( (\forall \mathcal{L})(\forall \mathcal{M})(\text{outgoing}(\mathcal{L}, \mathcal{M}) \rightarrow \text{next ch}(\mathcal{M}, \mathcal{L})) \).

Fig. 2. Rules in the sr-system.

Incoming data \( M \) happen at the same moment in time. Rules (r10) and (r11) can be explained in the same way. Rules (r12) and (r13) are involved in the communication, and their meanings are self-explanatory.

The initial conditions are given based on the actual states of the system. As an example, we may have the following initial conditions:

(i1) first \( \text{action}_{p1}(\text{pick}) \).

(i2) first \( \text{action}_{p2}(\text{receive}) \).

(i3) first \( \text{ch}(\mathcal{A}) \).

In summary, we have obtained a formal specification \( \mathcal{S} \mathcal{P} \mathcal{S}_{sr} = (\mathcal{V}, \mathcal{P}, \mathcal{ck}, \mathcal{R}, \mathcal{I}) \) of the sr-system, where

\( \mathcal{V} = \{ X, Y, M, W, Z, L \} \)

\( \mathcal{P} = \{ \text{state}_{p1}, \text{state}_{p2}, \text{action}_{p1}, \text{action}_{p2}, \text{ch}, \text{head}, \text{equal}, \text{incoming}, \text{outgoing} \} \).

\( \mathcal{ck} = \{ \text{state}_{p1} \mapsto \mathcal{ck}_1, \text{action}_{p1} \mapsto \mathcal{ck}_1, \text{state}_{p2} \mapsto \mathcal{ck}_2, \text{action}_{p2} \mapsto \mathcal{ck}_2, \text{ch} \mapsto gck, \text{head} \mapsto gck, \text{equal} \mapsto gck, \text{incoming} \mapsto gck, \text{outgoing} \mapsto gck \} \) where \( gck = \langle 0, 1, 2, 3, \ldots \rangle, \mathcal{ck}_1 = \langle 0, 2, 4, 6, \ldots \rangle \) and \( \mathcal{ck}_2 = \langle 1, 3, 5, 6, \ldots \rangle \)

\( \mathcal{R} = \{ r1, r2, \ldots , r13 \} \)

\( \mathcal{I} = \{ i1, i2, i3 \} \)

Usually, to obtain a consistent formal specification, a set of integrity constraints is needed [26]. The set of constraints provide requirements to each component of a formal specification. For example, for the clock assignment \( \mathcal{ck} \), we have the requirement that there are no predicates with which two different local clocks are associated. These
constraints can be used to check whether the knowledge obtained from any resource is consistent when constructing a formal specification.

5. Expressing timing properties in TLC

To describe reactive systems, we need an appropriate approach to specify their timing properties. In our framework, the fact that a reactive system satisfies a certain timing property is represented as a deducibility relation between the formal specification of the system and a number of formulas representing the property. In this section, we first give the notation of deducibility relations, then discuss the representation of timing properties of reactive systems such as safety, liveness and response properties. These properties are discussed in the following sections for the sr-system in more detail.

5.1. Deducibility relations, partial order

Given a reactive system $S$, we use the notation $\mathcal{PS}_S$ to denote the formal specification of $S$. We now give the following definitions:

**Definition 13.** Let $S$ be a reactive system and $A$ a formula. If $A$ can be derived from $\mathcal{PS}_S$, then we say that the system $S$ satisfies the deducibility relation denoted as follows:

$$\mathcal{PS}_S \vdash A$$

In particular, when $A$ is an event, we say that $A$ occurs in the system or $A$ is an event of the system.

**Definition 14.** Let $S$ be a reactive system and both $A$ and $B$ formulas. If $B$ can be derived from $\mathcal{PS}_S \cup \{A\}$, then we say that the system $S$ satisfies the deducibility relation denoted as follows:

$$\mathcal{PS}_S, A \vdash B$$

In particular, when $A$ and $B$ are both events, we say that $B$ occurs in the system if $A$ occurs in the system or $B$ is an event of the system if $A$ too is an event of the system.

In a reactive system, the notion of causality between events is essential in the formulation of certain timing properties of the system. Two events are constrained to occur in a certain order only if the occurrence of the first may affect the outcome of the second. One event may affect the other because the two events are of the same process, and thus may access the same local state, or because the two events are of different processes and they correspond to the exchange of a message. Intuitively, we need to consider two cases as follows:

- Two events are defined on the same local clock and the current time of an event is less than the current time of the other event. In this case, the occurrence of the first event could affect the outcome of the second.
Two events are associated with different local clocks. In this case, their order may depend on the exchange of a message. In our asynchronous (buffered) message-passing model, an event “send” should occur before the corresponding event “receive”; and any event, that occurs before the event “send”, must occur before any event, that occurs after the event “receive”.

We assume that send(M,L) and receive(M,L) are the predicates related to a pair of processes, the sending process \( p_1 \) and receiving process \( p_2 \), respectively. Here \( M \) is a message, and \( L \) the list of elements of the channel which connects the two processes.

Let \( S \) be a reactive system, \( \mathcal{P} \mathcal{S} = (\forall, \mathcal{P}, ck, \mathcal{R}, \mathcal{I}) \) its formal specification, and \( \prec \) the partial order relation over the set of all atomic events which can be derived from \( \mathcal{P} \mathcal{S} \). Then, by the definition of \( \prec \), we have the following facts:

1. Let
   \[ e_1 = \text{first next}[k_1]p(e_1,\ldots,e_n) \text{ and } e_2 = \text{first next}[k_2]p(s_1,\ldots,s_n) , \]
   where \( p \) is an \( n \)-ary predicate symbol. Then, when \( k_1 < k_2 \), we have that \( e_1 \prec e_2 \).

2. Let
   \[ e_1 = \text{first next}[k] \text{ send}(M,L) \text{ and } e_2 = \text{first next}[l] \text{ receive}(M,L') . \]
   Then \( e_1 \prec e_2 \).

In the analysis of reactive systems, by using the relation \( \prec \), we can effectively capture the intuitive notion of “cause-and-effect”. We have that \( e \prec e' \) if and only if \( e \) causally precedes \( e' \). The only conclusion that can be drawn from \( e \prec e' \) is that the occurrence of \( e' \) and its outcome may have been influenced by the event \( e \). Certain events of the global history may not be causally related, that is, it is possible that for some \( e \) and \( e' \), neither \( e < e' \) nor \( e' < e \). These events are called concurrent.

5.2. Time-forward systems

In the formal specification of a reactive system, it may be required that only future events are influenced by past events. Below, we introduce the notion of a time-forward system in which time-dependencies introduced through the use of temporal operators always satisfy the above mentioned requirement.

For any formula, all superfluous applications of temporal operators can be eliminated and temporal operators can be distributed to each pure atom by axioms. Thus, without loss of generality, we can assume that each formula as a rule or an initial condition in a specification has a canonical form of \( A \rightarrow B \), where \( A \) may be empty. Now we give the following definition:

**Definition 15.** Let \( S \) be a reactive system and \( \mathcal{P} \mathcal{S} \) its formal specification. If in any given rule or initial condition of the form \( A \rightarrow B \), there is no pure atom in \( A \) such that the number of next’s applied to the atom is greater than the number of next’s applied to any pure atom in \( B \), then we say that the formal specification \( \mathcal{P} \mathcal{S} \) is time-forward.
Obviously, the formal specification for the rs-system proposed in Section 3 is time-forward. For such specifications, we have

**Lemma 8.** Let $S$ be a reactive system and $\mathcal{S} \mathcal{P} \mathcal{B}_S$ is a time-forward specification. We assume that $e_1 = \text{first next}[n] A$ and $e_2 = \text{first next}[m] B$ be atomic events and $\mathcal{S} \mathcal{P} \mathcal{B}_S \vdash e_2$ but $\mathcal{S} \mathcal{P} \mathcal{B}_S, e_1 \vdash e_2$. Then

1. When $ck_{e_1} = ck_{e_2}$, we have that $n = m$ or $e_1 \prec e_2$, and
2. When $ck_{e_1} \neq ck_{e_2}$, we have that $e_1 \prec e_2$.

**Proof.** Case 1 is straightforward. Considering that the specification $\mathcal{S} \mathcal{P} \mathcal{B}_S$ is time-forward, and in a deduction procedure for deriving a conclusion from a combination of several antecedents, there are no rules which may cause that the current time of some formula as an antecedent is greater than the current time of that formula as the conclusion. Thus, when $ck_{e_1} = ck_{e_2}$, we must have $n \leq m$, which directly leads to the conclusion that $n = m$ or $e_1 \prec e_2$.

As for case 2, the deduction procedure for proving $e_2$ must involve one or more pairs of a send event and its corresponding receive event. If only one such a pair, say $e_s$ and $e_r$, is involved in the proof procedure, then, obviously, $e_1$ and $e_s$ should have the same local clock $ck_{e_1}$, and $e_2$ and $e_r$ have the same clock $ck_{e_2}$. On the other hand, if $e_1 \neq e_s$, we must have $e_1 \prec e_s$; similarly, if $e_2 \neq e_r$, we must have $e_r \prec e_2$. Therefore, from the fact that $e_s \prec e_r$, we have that $e_1 \prec e_2$. When more than one pair of a send event and its corresponding receive event are involved in the procedure for proving $e_2$, we can complete the proof by induction on the number of pairs.

We can also discuss the relation $\prec$ over the set of all events for a system. The ordering relation between given two events can be determined based on the atomic events involved in both the events.

### 5.3. Safety properties

Let $S$ be a reactive system, then a safety property of $S$ may be expressed in the following form:

$\mathcal{S} \mathcal{P} \mathcal{B}_S \vdash \Box A$

where $A$ is a TLC formula. It means that the system satisfies $A$ at all moments in time on the local clock $ck_A$, or simply, $A$ always holds (related to the local clock $ck_A$). By the definition of $\Box$, we can show that the safety property is equivalent to the following assertion:

$\mathcal{S} \mathcal{P} \mathcal{B}_S \vdash \text{first next} \left[ \text{rank}(t, ck_A) \right] A$, for all $t \in ck_A$.

That is, a safety property can also be expressed in the above form.

Suppose that we have the safety property as follows:

$\mathcal{S} \mathcal{P} \mathcal{B}_S \vdash \text{first next}[n] (A \rightarrow \text{next}[i] B)$, for any $n \in \omega$.  

where $i$ is a defined natural number and $c_{k_A,B}$ is, of course, assumed to be an infinite clock. When $c_{k_A} = c_{k_B}$, the safety property can be expressed in the following form:

$$\mathcal{P}_B \text{ first next}[n] A \vdash \text{ first next}[n+i] B \text{ for any } n \in \omega.$$ 

More generally, we may have a safety property of the following form: For any $n \in \omega$, there exists $m \in \omega$, such that

$$\mathcal{P}_B \text{ first next}[n] A \vdash \text{ first next}[m] B$$ 

where $c_{k_A}$ and $c_{k_B}$ may be different. Here $n$ may or may not be less than $m$, but the event $\text{ first next}(n)$ $A$ should occur before the occurrence of the event $\text{ first next}(m)$ $B$ if they are not concurrent events. That is, we should have that $c_{k_A}^{(n)} < c_{k_B}^{(m)}$.

We may also have the safety properties of the following form: For a defined natural number $i$,

$$\mathcal{P}_B \vdash \Box \text{ next }[i] A$$

which means that $A$ is true at all moments in time on the local clock $c_{k_A}$ from the $i$th next moment after the initial moment in time. Its equivalent expression is as follows:

$$\mathcal{P}_B \vdash \Box \text{ first next}[\text{ rank}(t, c_{k_A})] A$$

for all $t \in c_{k_A}$ such that $\text{ rank}(t, c_{k_A}) \geq i$.

A bounded invariance property can be expressed in the form:

$$\mathcal{P}_B \vdash \bigwedge_{i=0}^{n} \text{ next }[i] A.$$ 

The property states that $A$ is true until the $n$th next moment in time from the current moment.

An example of a bounded invariance property of a given reactive system $S$ is given as follows: Assuming that a customer $X$ is waiting for service in a queue (of several items) and the time for each service item is one unit of time (based on a given local clock), we may have a property as follows:

- **When $X$ becomes the first customer in the queue, he/she will get $k$ consecutive items of service.**

Then, the property may be expressed as

$$\mathcal{P}_B, \text{ first next}[n] \text{ customer}_f(X) \vdash \text{ first next}[n] \left( \bigwedge_{i=0}^{k} \text{ next }[i] \text{ get service}(X) \right)$$

where $\text{ customer}_f(X)$ means that $X$ is the first customer in the queue and $\text{ get service}(X)$ $X$ gets a service.

### 5.4. Liveness properties

Let $S$ be a reactive system, then a liveness property of $S$ may be expressed in the following form:

$$\mathcal{P}_B \vdash \Diamond A$$
where $A$ is a TLC formula. It means that the system satisfies $A$ at some moment in
time on the local clock $ck_A$, or simply, $A$ holds at some moment in time (related to
the local clock $ck_A$).

By the definition of $\Diamond$, we can show that the liveness property is equivalent to the
following assertion:

$$SP\mathcal{E}_f \vdash \text{first next}[\text{rank}(t,ck_A)] A\text{, for some } t \in ck_A.$$

Suppose that we have the liveness property as follows:

$$SP\mathcal{E}_f \vdash \text{first next}[n] (A \to \text{next}[i] B)\text{, for some } n \in \omega.$$

In the property, $i$ is a defined natural number and $ck_{A,B}$ is assumed to be an infinite
clock.

When $ck_A = ck_B$, the property can be expressed in the following form:

$$SP\mathcal{E}_f \text{ first next}[n] A \vdash \text{first next}[n + i] B\text{ for some } n \in \omega.$$

As in the discussion of safety properties, we may have a liveness property of a more
general form: There is a natural number $n \in \omega$, such that for some $m \in \omega$,

$$SP\mathcal{E}_f, \text{ first next}[n] A \vdash \text{first next}[m] B$$

where $ck_A$ and $ck_B$ may be different.

We may also have a liveness (response) property of the following form: for a defined
natural number $i$,

$$SP\mathcal{E}_f \vdash \Diamond \text{next}[i] A$$

which means that $A$ is true at some moment in time on the local clock $ck_A$ from the $i$th
next moment after the initial moment in time. Its equivalent expression is as follows:

$$SP\mathcal{E}_f \vdash \Box \text{first next}[\text{rank}(t,ck_A)] A$$

for some $t \in ck_A$ such that $\text{rank}(t,ck_A) \geq i$.

A bounded response property can be expressed in the form:

$$SP\mathcal{E}_f \vdash \bigvee_{i=0}^{n} \text{next}[i] A.$$

The property states that $A$ is true at at least one moment from the current moment in
time to the $n$th next moment. More examples of response properties are given in the
next section.

6. Timing properties of the $sr$-system

This section discusses some safety and liveness properties of the $sr$-system. In order
to formalize the timing properties of the $sr$-system, we first define two new auxiliary
predicates as follows:

$$\text{send}(M, L) \overset{\text{def}}{=} \text{state}_p(M) \land \text{ch}(L) \land \text{action}_p(\text{send}).$$
receive(M,L) \equiv \text{ch}(L) \land \text{head}(M,L) \land \text{action}_p^2(\text{receive}).

For the new auxiliary predicates, we have that $ck(\text{send}) = ck_1$ and $ck(\text{receive}) = ck_2$.

We now state a typical response property, which a given reactive system $S$ should satisfy, as follows:

- Any message $M$ sent out by a sending process of the system must be received by the corresponding receiving process in a permitted time interval, say a time period not exceeding 10 units of time (assuming, based on the global clock).

In our framework, using the predicates $\text{send}(M,L)$ and $\text{receive}(M,L)$, such a property may be expressed as follows.

\[
\text{first next}(n) \text{send}(M,L) \vdash \text{first next}(m)(\forall i \in [n+1, n+10] \text{receive}(M,L'))
\]

Here, if $n = \text{rank}(t_1, ck_1)$, $m = \text{rank}(t', ck_2)$, $m + k = \text{rank}(t'', ck_B)$, then we should have that $t' = \text{min}\{t \mid t \in ck_B \text{ and } (t > t_1)\}$ and $t'' \leq t_1 + 10$.

We now give some other important timing properties of the $sr$-system.

**Property 1.** The $sr$-system has the following safety properties related with the action "send": For any natural number $k \in \omega$,

\[
\begin{align*}
\mathcal{P}_{B_{sr}}, \text{first next}[k] \text{action}_p^1(\text{send}) & \vdash \text{first next}[k+2] \text{action}_p^1(\text{send}). \\
\mathcal{P}_{B_{sr}}, \text{first next}[k] \neg\text{action}_p^1(\text{send}) & \vdash \text{first next}[k+1] \text{action}_p^1(\text{send}).
\end{align*}
\]

**Property 2.** The $sr$-system has the following safety properties related with the action "receive": For any natural number $k \geq 1$,

\[
\begin{align*}
\mathcal{P}_{B_{sr}}, \text{first next}[k] \text{action}_p^2(\text{receive}) & \vdash \text{first next}[k+2] \text{action}_p^2(\text{receive}). \\
\mathcal{P}_{B_{sr}}, \text{first next}[k] \neg\text{action}_p^2(\text{receive}) & \vdash \text{first next}[k+1] \text{action}_p^2(\text{receive}).
\end{align*}
\]

Properties 1 and 2 are actually involved in exact time, therefore, they can also be viewed as real-time properties. Intuitively, the meaning of Property 1 is that, at any moment in time (based on the local clock $ck_1$ of the predicate $\text{action}_p^1$), if process $p_1$ is executing a "send" action, then the next "send" action must be executed at the second next moment. Similarly, Property 2 indicates that, after the initial moment in time, at any moment in time (based on the local clock $ck_2$ of the predicate $\text{action}_p^2$), if process $p_2$ is executing a "receive" action, then the next "receive" action must be executed at the second next moment.

If the events involved in a property are associated with the same local clock, then the property is local. For example, Property 1 above is local. A property which is not
local is called a global property. For example, we have the following global eventuality property:

**Property 3.** Let \( \text{first next}[k] \) \( \text{send}(M,L) \) be an event of the sr-system, where \( k \in \omega \) and \( M \) is any message. Then there exists an event \( \text{first next} [l] \) \( \text{receive}(M, L') \), where \( l > 0 \) and \( L' \) is a list (different from \( L \)), such that

\[
\mathcal{P}_r', \text{first next}[k] \) \( \text{send}(M,L) \vdash \text{first next} [l] \) \( \text{receive}(M, L').
\]

The property is involved in two processes, the sending process and the receiving process, which are associated with different local clocks. It indicates the fact that:

- Any message \( M \) sent by process \( p_1 \) must be received by process \( p_2 \) at some moment in time in the future.

Note that, in the property, \( k \) may be not less than \( l \) because the local clocks of send and receive are different, but the current time of the formula \( \text{first next}[k] \) \( \text{send}(M,L) \) must be less than the current time of \( \text{first next} [l] \) \( \text{receive}(M,L) \).

Another example of a global property is the First-In-First-Out (FIFO) delivery rule, which the communication from a process to another process may satisfy. The rule can be formally represented as follows:

**Property 4.** Suppose both the events \( e_r = \text{first next}[k_1] \) \( \text{send}(M_1,L_1) \) and \( e'_r = \text{first next}[k_2] \) \( \text{send}(M_2,L_2) \) occur in the sr-system, then there exist events

\[
e_r = \text{first next}[l_1] \) \( \text{receive}(M_1,L_1') \) and \\
e'_r = \text{first next}[l_2] \) \( \text{receive}(M_2,L_2') \),
\]

such that if \( e_r < e'_r \), then \( e_r < e'_r \).

In our send-receive system, the maximum capacity of channel \( ch \), which connects the processes \( p_1 \) and \( p_2 \), is \( d \). So, in order to guarantee that no messages will be lost, the specification must satisfy a response property. This property asserts that a message, say the 1st message, sent by process \( p_1 \) must be received before the \( (d+1) \)th message is sent out. Note that, if the first message \( M_0 \) is sent out at the current moment in time, then the second one will be sent out at the second next moment, the third one will be sent out at the fourth next moment, and so on. Thus, if the event “sending first message \( M_0 \)” is denoted by the formula \( \text{first next}[k] \) \( \text{send}(M_0,L_0) \), then the event “sending \( (d+1) \)th message \( M_k \)” will be denoted by \( \text{first next}[k+2d] \) \( \text{send}(M_k,L_k) \).

Therefore, we have the following property:

**Property 5.** Let

\[
e_0 = \text{first next}[k] \) \( \text{send}(M_0,L_0) \) and \\
e_k = \text{first next}[k+2d] \) \( \text{send}(M_k,L_k),
\]

be events of the sr-system, then there exists an event \( e = \text{first next}[l] \) \( \text{receive}(M_0,L) \), such that \( \mathcal{P}_r' \), \( e_0 \vdash e \) and \( c_{k+2d}^{(l)} < c_{k}^{(l+2d)} \).
This property is actually a bounded response property. It can also be expressed in the following form:

\[ \mathcal{P}\Phi_{\text{sr}}, \text{first next } [k] \text{ send}(M_0, L_0) \vdash \text{first next } [m](\bigvee_{i=0}^{\infty} \text{next}[i] \text{ receive}(M_0, L)) \]

where \( m \) and \( r \) satisfy the condition that \( c_{k_2}^{(m+r)} = \min\{t \mid (t \in c_{k_2}) \text{ and } (t > c_{k_1}^{(k)})\} \) and \( c_{k_2}^{(m+r)} \leq c_{k_1}^{(k+2d)} \).

Note that the clock assignment is a very important component to the formal specification of a reactive system. An inappropriate clock assignment may cause a system to fail to satisfy some requirements. For example, in the sr-system, if the clock \( c_{k_2} = (1, 5, 9, 13, \ldots) \) and there is no change for \( c_{k_1} \), then the system will not satisfy the response Property 5. Based on the logic TLC, the designer of a system is free to choose local clocks for predicates. However, when some requirements such as timing properties which the system should meet in any proposed implementation have been given, the clock assignment should be given based on these requirements.

7. Verification by local reasoning

Local reasoning can be used when a timing property involves a set of events, all of which are defined on the same local clock. In other words, local reasoning allows us to prove the property without considering clocks, so that the proof procedure for verifying a timing property is simplified. It plays an important rôle for reasoning about the local properties as well as the global properties in reactive systems.

7.1. Closed local subsets

We first define the notion of a closed local subset.

**Definition 16 (Liu and Orgun [27])**. Let \( \mathcal{P}\Phi = (\forall, \mathcal{P}, c_k, \mathcal{R}, \mathcal{I}) \) be the specification of a given reactive system and \( \mathcal{L} \subseteq \mathcal{P} \). If, for any \( q \in \mathcal{L} \), \( c_k(q) = c_{k_i} \), where \( c_{k_i} \) is a certain local clock, then the union of a subset of \( \mathcal{R} \), named \( \mathcal{R}(\mathcal{L}) \), and a subset of \( \mathcal{I} \), named \( \mathcal{I}(\mathcal{L}) \), is called the closed local subset with \( \mathcal{L} \) if it satisfies the following conditions:

- If \( A \in \mathcal{R}(\mathcal{L}) \cup \mathcal{I}(\mathcal{L}) \) and \( q \) is a predicate symbol occurring in \( A \), then \( q \in \mathcal{L} \).
- If \( A \in \mathcal{R} \cup \mathcal{I} \), \( q \in \mathcal{L} \) and \( q \) is contained in the conclusion of \( A \), then \( A \in \mathcal{R}(\mathcal{L}) \cup \mathcal{I}(\mathcal{L}) \).

The first condition of this definition guarantees that a proof over \( \mathcal{R}(\mathcal{L}) \cup \mathcal{I}(\mathcal{L}) \) is closed over the predicate set \( \mathcal{L} \), i.e., at any step of a proof, no formulas containing predicates other than those in \( \mathcal{L} \) are generated if we only use the formulas in \( \mathcal{R}(\mathcal{L}) \cup \mathcal{I}(\mathcal{L}) \).

The second condition of the definition guarantees that a predicate in \( \mathcal{L} \) may not be implied by any formulas other than those in \( \mathcal{R}(\mathcal{L}) \cup \mathcal{I}(\mathcal{L}) \), i.e., for deriving a formula constructed by predicates in \( \mathcal{L} \), all formulas involved in the reasoning procedure belong to the set \( \mathcal{R}(\mathcal{L}) \cup \mathcal{I}(\mathcal{L}) \). Thus, we have the following result:
Lemma 9. Let $\mathcal{S}_L = (V, P, c, R, I)$ be the specification of a given reactive system. Let $R(2) \cup I(2)$ be a closed local subset with $2$, and $\Lambda$ be a formula in TLC and for any predicate $q$ appearing in $\Lambda$, we have $q \in 2$. Then

$\mathcal{S}_L \vdash \Lambda$ if and only if $\mathcal{S}_L(2) \vdash \Lambda$

where $\mathcal{S}_L(2) = (V', \mathcal{V}, \{c_{ki}\}, \mathcal{R}(2), \mathcal{I}(2))$, $V'$ is a set of $V$ which only contains all variables appearing in $2$, $c_{ki}$ is the local clock associated with the predicates in $2$.

According to the definition of a closed local subset, the correctness of the lemma is straightforward and we omit its proof.

7.2. Proving timing properties

In general, verifying a local timing property of a system by local reasoning includes the following steps:
- to find the set of predicates involved in the property and check if their local clocks are the same;
- to find the closed local subset with the set of predicates found at step 1;
- to prove the property based on the closed local subset.

As an example of the use of local reasoning for the verification of timing properties of reactive systems, we now give a formal proof for Property 1.

Proof of Property 1. Let $2 = \{\text{action}_p_1\}$. It is obvious that $\{r_1, r_2, r_3\} \cup \{i_1\}$ is the closed local subset with $2$. Let

$\mathcal{S}_L(2) = (\{w\}, \{\text{action}_p_1\}, \{c_k_1\}, \{r_1, r_2, r_3\}, \{i_1\})$.

Then, in the specification, we can obtain the following proof:

(1) first next(k) action_p_1(send). \hspace{1cm} (assumption)
(2) action_p_1(send) \rightarrow next action_p_1(pick). \hspace{1cm} (r2)
(3) action_p_1(pick) \rightarrow next action_p_1(send). \hspace{1cm} (r1)
(4) next action_p_1(pick) \rightarrow next[2] action_p_1(send). \hspace{1cm} (from (3), by R3)
(5) action_p_1(send) \rightarrow next[2] action_p_1(send). \hspace{1cm} (from (2) & (4))
(6) first next[k] (action_p_1(send) \rightarrow next[2] action_p_1(send)). \hspace{1cm} (from (5), by R3 & R2)
(7) first next[k + 2] action_p_1(send). \hspace{1cm} (from (1) & (6))

Therefore, we have that

$\mathcal{S}_L_{sr}(2)$, first next[k] action_p_1(send) \vdash first next[k + 2] action_p_1(send).

Then, according to Lemma 9, we have that

$\mathcal{S}_L_{sr}$, first next[k] action_p_1(send) \vdash first next[k + 2] action_p_1(send).
To prove the second part of **Property 1**, we have that

(8) \( \text{first next}[k] \ (\text{action}_{P_1}(\text{send}) \rightarrow \text{next action}_{P_1}(\text{pick})). \) (from 2, by R1, R2)

(9) \( \text{first next}[k + 1] \ \text{action}_{P_1}(\text{pick}). \) (from (1) & (8), by Modus Ponens)

(10) \( (\neg \text{action}_{P_1}(\text{send}) \lor \neg \text{action}_{P_1}(\text{pick})). \) (r3)

(11) \( \text{first next}[k + 1] \ (\neg \text{action}_{P_1}(\text{send}) \lor \neg \text{action}_{P_1}(\text{pick})). \) (from (10), by R2, R3)

(12) \( \text{first next}[k + 1] \ \neg \text{action}_{P_1}(\text{send}). \) (from (9) & (11))

This is what we need to show. □

In order to prove a property local to a process, we may only need to consider the local clock of the process. In general, if we can find a closed local subset from the formal specification of the system, we may prove the property only by local reasoning. However, there are some local properties, for which we cannot find such a closed local subset. Therefore, we can not prove such a property just by local reasoning. For example, **Property 2** is also a local property, in which the only predicate is \( \text{action}_{P_2}. \) Its local clock is \( c_{k_2}, \) but there is no closed local subset with the predicate. Actually, the proof of **Property 2** involves in the predicates \( \text{ch} \) and \( \text{equal}, \) which are associated with the global clock \( g_{ck}. \) Local reasoning can be used not only to prove local timing properties of reactive systems, but also to prove some global properties. In the proof procedure of a (local or global) property, there may exist some intermediate results which can be proved by local reasoning. If that is the case, then the overall proof will be simplified as we do not have to consider the full proof system for proving those intermediate results.

8. Verification using fixed-time rules

In reactive systems, there are many properties which are related with communications between processes. Such properties may involve several formulas which are associated with different local clocks, and cannot be verified by only using local reasoning. To verify such timing properties, we need to use fixed-time rules. Fixed-time rules can be used to deal with communicating processes of a system. In this section, we demonstrate the verification of timing properties of a reactive system using fixed-time rules through a practical example.

We now give a formal proof for **Property 3** of the \( sr\)-system.

**Proof.** According to the formal specification, by the axioms, inference rules and fixed-time rules for TLC, we have the following proof. It should be kept in mind that events are ground, fixed-time formulas.

(1) \( \text{first next}[k] \ \text{send}(M,L). \) (assumption)
(2) first next \[k\] (state_p1(M) \land ch(L) \land action_p1(send)). \hspace{0.5cm} \text{(definition)}

(3) state_p1(M) \land ch(L) \land action_p1(send) \rightarrow incoming(L,M).

\[r9, \text{substitution}\]

(4) first next \[k\] (state_p1(M) \land ch(L) \land action_p1(send) \rightarrow \text{incoming}(L,M)). \hspace{0.5cm} \text{(from (3), by R2, R3)}

By F\rightarrow_1, from (2) and (4) and considering the concrete clocks associated with these formulas, we have

(5) first next \[2k\] incoming(L,M).

Thus, we have

\[\text{By F-1, from (2) and (4) and considering the concrete clocks associated with these formulas, we have}\]

(6) incoming(L,M) \rightarrow next ch(L*M).

\[r12, \text{substitution}\]

(7) first next \[2k\] (incoming(L,M) \rightarrow next ch(L*M)). \hspace{0.5cm} ((6), by R3, R2)

(8) first next \[2k + 1\] ch(L*M).

\[\text{from (5) \& (7), by F_1}\]

Let \(|L|\) denote the length of \(L\), i.e., the number of elements contained in \(L\). Below, we want to show that first next \([l]\) receive(M,L') can be derived from (8) by induction on \(|L|\). For convenience, we prove it in the more general case, that is, we write (8) as

(8') first next \([2k + 1]\) ch(L*M*L_1)

from which we will directly derive the conclusion needed. Firstly, we assume that \(|L| = 0\), i.e., \(L = \lambda\). Then we have

(9) first next \((2k + 1)\) head(M,L*M*L_1).

From rules (r5)-(r7), the initial condition (i2) and substitution, we have

first next \([k]\) action_p2(Z),

where Z=receive or process. If Z=receive, then we have

(10) first next \([k]\) action_p2(receive).

Thus, from (8'), (9) and (10), and by the fixed-time rule (F\land_1), we have

(11) first next \([k]\) (ch(L*M*L_1) \land head(M,L*M*L_1) \land action_p2(receive)).

So, according to the definition, we have

(12) first next \((k)\) receive(M,L*M*L_1).

Let \(L'=L*M*L_1\) and \(l=k\), we therefore have that

(13) first next \([l]\) receive(M,L').

If Z=process, then we have

(14) first next \([k + 1]\) action_p2(receive).
In this case, from (8') and (9), we may obtain that

\[(15) \text{first next}[2k + 3] \text{ch}(L*M*Lz).\]
\[(16) \text{first next}[2k + 3] \text{head}(M,L*M*Lz).\]

Note that, because of the effect of the sending process, \(Lz\) and \(Li\) may be different. Thus, similarly, from (14), (15) and (16) we can also obtain that

\[(17) \text{first next}[\lambda] \text{receive}(M,L').\]

when we let \(L'=L*M*Lz\) and \(\lambda = k + 1\).

Therefore, according to the above facts, we have that, when \(|L| = 0\), Property 3 holds. Secondly, assume that, when \(|L| = n\), the property holds. We now consider the case when \(|L| = n + 1\). Let the first element of \(L\) in (8') to be \(M'\), then we have

\[(18) \text{first next}[2k + 1] \text{head}(M',L*M*L_1).\]

Thus, according to the proof of the base step, there exists \(s > 0\) such that we have

\[(19) \text{first next}[s] \text{receive}(M',L*M*Lz).\]

Then, by rule (r10) and (r13) and substitution, we have

\[(20) \text{first next}[2s + 1] \text{ch}(M'/L*M*Lz).\]

Thus, according to the induction assumption, from (20), because \(|M'/L| = n\), we have that there exists \(l > 0\) such that we have

\[(21) \text{first next}[\lambda] \text{receive}(M,L').\]

That is what we want to show. \(\square\)

The proofs of Properties 2, 4 and 5 can be obtained in the same way. Therefore no details are given.

The above proof shows that the verification of timing properties of a reactive system can be automatically obtained, based on its formal specification, from the axioms, inference rules and the fixed-time rules. However, we have to note that such proofs are based not only on the sets of rules and initial conditions of the system, but also on the clock assignment. We often need to refer to the clock assignment to determine the form of a formula derived through computing ranks of moments in time on local clocks.

9. Concluding remarks

TLC is particularly suitable for describing the behavior of those systems that may involve several parallel processes, and in which the program and its environment act concurrently. In this paper, we have extended the formal specification of reactive systems proposed in [27], and presented a corresponding verification framework for proving their timing properties. In our logic, a reactive system and its corresponding proper-
ties are all represented by TLC formulas, and the properties can therefore be directly deduced from the specification of the system.

The framework for verifying timing properties of reactive systems includes an application of local reasoning and deriving fixed-time rules from the proof system of TLC. Using the proof techniques discussed in this paper, global and local timing properties of a reactive system can be verified. Due to the representation the behavior of reactive systems and their corresponding properties in our framework, induction, in many cases, is also very useful for the verification procedure.

A complete study of the use of fixed-time rules is necessary. A formal method to use such rules for verifying timing properties, in particular, for verifying global timing properties, may also be needed. Future work also includes providing a practical automated verification method for timing properties of realistic reactive systems.

Acknowledgements

The work presented in this article has been supported in part by an Australian Research Council (ARC) Grant. C. Liu was supported by an Australian Postgraduate Award (APA) and an MPCE Postgraduate Research Scholarship at Macquarie University while working on this project. Thanks are also due to two anonymous referees for their valuable comments and suggestions.

References