Front.Comput.Sci.
DOI

RESEARCH ARTICLE

A survey on temporal logics for specifying and verifying real-time systems

Savas KONUR (✉)

Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK

Abstract Over the last two decades, there has been an extensive study of logical formalisms on specifying and verifying real-time systems. Temporal logics have been an important research subject within this direction. Although numerous logics have been introduced for formal specification of real-time and complex systems, an up to date survey of these logics does not exist in the literature. In this paper we analyse various temporal formalisms introduced for specification, including propositional/first-order linear temporal logics, branching temporal logics, interval temporal logics, real-time temporal logics and probabilistic temporal logics. We give decidability, axiomatizability, expressiveness, model checking results for each logic analysed. We also provide a comparison of features of the temporal logics discussed.

Keywords propositional temporal logics, first-order linear temporal logics, branching temporal logics, interval temporal logics, real-time temporal logics, probabilistic temporal logics, decidability, model checking, expressiveness

1 Introduction

Temporal logics are formal frameworks which describe statements whose truth values change over time. Whereas classical logics do not include a time element, temporal logics characterize state changes which depend on time. This makes temporal logics a richer notation than classical logics.

Temporal logics have been extensively used in the specification of various systems, such as real-time and control systems, for more than two decades. They provide a mathematical foundation to formally analyse these systems. Many industrial applications and case studies proved the usability of temporal logics within this context.

In general, temporal logics have been introduced for specific types of problems. The general trade-off is between the complexity and expressiveness. In certain applications simple logics are preferred to the complex ones [1]. Complex logics are generally difficult to deal with practically.

Numerous logics have been introduced for the formal specification of real-time and complex systems, and various aspects of logics have been studied. Some surveys (such as [1–7]) can be found in the literature but these mainly concentrate on specific formal systems over specific structures of time, and they do not cover a broad range of logics. Also, numerous new results have been published recently; an up to date survey of these does not exist in the literature.

In this paper we attempt to fill this gap by describing a broad spectrum of temporal logics, and outlining main and recent developments in this field. All these logics are different in terms of 'expressiveness', 'order', 'time metric', 'temporal modalities' and 'time model'.

In this paper we survey the following aspects: 'basic temporal framework', 'real-time' and 'probability'. The real-time aspect of temporal logics is important to express timing requirements of real-time systems. The probabilistic aspect is needed in order to reason about systems which include uncertainty and probabilistic assumptions.

This paper contributes in the following directions: An up to date survey including the recent results is conducted. Also, to our best knowledge, no survey of probabilistic temporal logics has been done so far. This paper also covers probabilistic logics and attempts to contribute in this direction as

This is a pre-copy-editing, author-produced PDF. The definitive publisher-authenticated version is available online, DOI: 10.1007/s11704-013-2195-2

E-mail: konur@liverpool.ac.uk

well. In addition, the paper addresses a very wide range of temporal logics, which was not done in the surveys we mentioned above. Indeed, the work includes propositional/first-order linear temporal logics, branching temporal logics, interval temporal logics, real-time temporal logics and probabilistic temporal logics.

We will summarize important results on decidability, axiomatizability, expressiveness, model checking, etc. for each logic analysed, whenever possible, and will provide a comparison of features of the temporal logics discussed.

We believe that this survey will be very useful for researchers to have a wide picture of the spectrum and to find an up to date related works for important logics [1)].

2 Preliminaries

Temporal logics can be considered as extensions of classical propositional and first-order logic. In fact, propositional temporal logics are an extension of propositional logic with temporal operators. Similarly, first-order temporal logics are extension of first-order logic with temporal modalities. Temporal logics are also a special type of modal logics, where statements are evaluated on 'worlds' which represent time instants.

We can classify temporal logics based on several criteria. The common dimensions are 'propositional versus first-order', 'point-based versus interval-based', 'linear versus branching', 'discrete versus continuous', etc [1,8,9]. Below we discuss the most important criteria to classify temporal logics.

Point versus interval structures:

There are two structure types to model time in a temporal logic: *points (instants)* and *intervals*. A point structure $\mathbb{T}$ can be represented as $\langle T, <\rangle$, where T is a nonempty set of time points, and $<$ is a 'precedence' relation on T. Different temporal relationships can be described using different modal operators. Some logics include modal operators which can express quantification over time. However, a relationship between intervals is difficult to express using a point-based temporal logic [10].

Interval temporal logics are expressive, since these logics can express a relationship between two events, which are represented by intervals. Also, interval logics [11–17] have a simpler and neater syntax to define a relationship between intervals, which provides a higher level abstraction than a point-based logic when modeling a system. This makes interval logic formulas much simpler and more comprehensive than point-based logic formulas.

Some of the known interval operators are *meets*, *before*, and *during* [18], which denote the ordering of intervals[2)], the *chop* modality [19], which denotes combining two intervals, and *duration*, which denotes a length of an interval [6].

Interval structures can be considered in two ways: *(i)* intervals are 'primitive' objects *(ii)* intervals are composed from points. [20–22] consider intervals as primitive objects of time. [20] defines a 'period structure' as the tuple $\langle \mathbb{I}, \subseteq, \prec \rangle$, where $\mathbb{I}$ is a non-empty set of intervals, $\subseteq$ is a sub-interval relation, and $\prec$ is a precedence relation. One particular problem with this approach is that theoretical analyses are usually very difficult. Also, although it is very easy to define properties such as *linearity*, *density*, *discreteness*, *unboundedness* in a point-based logic, it is very difficult to define these properties in an interval logic where intervals are primitive objects.

[7, 17, 19] consider intervals as sets of points, where the time flow is assumed as 'a strict partial-ordering of time points'. Namely, an interval structure is defined as $\langle \mathbb{T}, \mathbb{I}(\mathbb{T}) \rangle$, where $\mathbb{T} = \langle T, < \rangle$ is a strict partial-ordering and $\mathbb{I}(\mathbb{T})$ is a set of intervals. The properties mentioned above can be defined in an interval logic where intervals are composed of time instants.

Here we give the historical development of interval-based temporal logics. The concept of time intervals was first studied by Walker [23], who considered a non-empty set of intervals, which is partially ordered. However, his work does not cover aspects of temporal logic in a general sense. In [24] philosophical aspects of an interval ontology was analysed. In [25] an interval tense logic was introduced. [26–32] studied interval logics within the natural language domain. It was argued that interval-based semantics are more convenient for human language and reasoning, and the interval-based approach is more suitable than the point-based approach for temporal constructions of natural language. [18, 33–35] studied event relations and interval ordering. The authors introduced so-called Allen's thirteen interval relations and worked on axiomatisation and representation of interval structures. Some further works on Allen's algebra were carried out by [14, 36]. Recently, [37] investigated the relation between

[1)] Note that in some instances we think it is more convenient to refer to the original text for clarification purposes. In the following, we will use quotation marks to use the text from the original sources.

[2)] We say that an interval J *meets* another interval J if the end point of I is the starting point of J, I *before* J if the end point of I is before the starting point of J, and I *during* J if the starting point of I is after the starting point of J and the end point of I is before the end point of J.

Allen's logic and LTL. Interval based-logics have been also applied to other fields in computer science. [38–40] worked on process logic, where intervals are used as representation of information. Another important work was the development of *interval temporal logic (ITL)*, and its application to design of hardware components [13, 41]. Since the development of ITL, numerous variations have been proposed; in particular, Duration Calculus [6] is an extension of interval temporal logic with "a calculus to specify and reason about properties of state durations".

Temporal Structure:

There are important properties regarding the time flow and temporal domain structure. Some properties are summarized below:

Assume $\langle T, <\rangle$ represents a temporal structure, where T is a nonempty set of time points, and $<$ is a 'precedence' relation on T. In a temporal logic the structure of time is *linear* if any two points can be compared. Mathematically, a strict partial-ordering is called linear if any two distinct points satisfy the condition: $\forall x, y : x < y \vee x = y \vee x > y$. This definition suggests that in linear temporal logics each time point is followed at most one successor.

Another class is the *branching-time structures*, where the underlying temporal structure is branching-like, and each point may have more than one successor point. The structure of time can be considered as a tree. A *tree* is a set of time points T ordered by a binary relation $<$ which satisfies the following requirements [42]:

- $\langle T, <\rangle$ is irreflexive;
- $\langle T, <\rangle$ is transitive;
- $\forall t, u, v \in T$ $u < t$ and $v < t \rightarrow u < v, u = v$ or $u > v$ (i.e. the past of any point is linear);
- $\forall x, y \in T, \exists z \in T$ such that $z < x$ and $z < y$ (i.e. $\langle T, <\rangle$ is connected);
- $z \in T$ is the *root* of $\langle T, <\rangle$ iff $\forall x \in T$ $z \leq x$.

One important characteristic of branching logics is that the syntax of these logics include path quantification which allows formulas to be evaluated over paths. However, linear temporal logics are restricted to only one path.

A temporal domain is *discrete* with respect to the precedence relation $<$ if each non-final point is followed by a successor point. This can be formulated as follows: $\forall x, y$ $(x < y \rightarrow \exists z\, (x < z \wedge \neg\exists u(x < u \wedge u < z)))$. The majority of temporal logics used for system specification are defined on discrete time, where points represent system states. A state sequence, as a result of a program execution, can be considered as isomorphic to the discrete series of positive integers.

A temporal domain is *dense* if, between any two distinct points, there is another point. This can be formally denoted $\forall x, y(x < y \rightarrow \exists z(x < z < y))$. Above we mentioned that a flow of discrete time can be represented as a set of positive integers. Similarly, a dense domain can be represented as a set of the positive real numbers. It is noteworthy to mention that there is a distinction between *density* and *continuity*: "A model of dense time is isomorphic to a dense series of rational numbers, meaning that there is always a rational number between any two rational numbers; whereas a model of *continuous* time is isomorphic to a continuous series of real numbers" [9].

A temporal domain is *bounded above* (*bounded below*) if the temporal domain is bounded in the future (past) time. This can be formulated as follows: $\exists x \neg \exists y(x < y)$ $(\exists x \neg \exists y(y < x))$. Similarly, a temporal domain is *unbounded above* (*unbounded below*) if each point has a successor (predecessor) point, which is formally denoted $\forall x \exists y(x < y)$ $(\forall y \exists x(y < x))$.

A temporal domain *is Dedekind complete* if all time point sets (non-empty) are bounded above, and they have a least upper bound.

Based on differences in temporal domain properties logics have different characteristics. For example, we can consider a temporal domain which is linear or branched; discrete or dense; finite/infinite in future and/or past, etc. All these choices result in different syntax, semantics, decidability and complexity.

Before we end this section, we provide a brief account on early historical development of temporal logic, mainly summarised from [4]:

The development of temporal logic goes back to the middle of the twentieth century. The major work was done by Prior, who developed a logical framework in order to formally analyse some natural language constructions. Prior dealt with a number of tense-logical systems, made important discoveries in modal logic, and classified various possible positions in relation to temporal and modal logic. He considered temporal instants as a special type of propositions. Using his *tense logic* [43–45] [3], he predicted thirty different tenses on certain given assumptions by using tense operators. The language of his tense logic contains the following modal operators (in addition to the the standard boolean operators): "sometime in the past", "sometime in the future", "always in

[3] A revised and expanded edition of [45] was published by Oxford University Press in 2003.

the past" and "always in the future".

Another important work of Prior was his contribution on the development of the *branching* time. Until 1940s, time models had been dominantly considered either as linear or circular. The idea of branching time was first proposed by Borges; but initial suggestions about branching time came from Saul Kripke. However, there was no proper formulation until Prior's work. He provided a model for branching time by considering the history as maximal linearly-ordered set of time instants.

After the introduction of the tense logic, many logicians have worked on it, e.g. [26, 31]. Among them Kamp's work was very influential. Kamp [46] introduced two new operators ("since" and "until") and proved that any other temporal operators can be defined in terms of these two operators provided that time is continuous.

One important extension to tense logic was the *metric tense logic* introduced by Prior [44, 45], where he considered numerical durations, such as "some future time within n".

Prior initially considered the tense logic within philosophical logic. He later concentrated on the mathematical issues. During the last two decades the relevance of temporal logic within various disciplines was realised. Especially, in 1970s the relevance to computer science was better understood. After this time temporal logic has been used in various fields of computer science ranging from natural language processing to formal aspects of real-time systems.

3 Propositional Temporal Logics

Propositional temporal logics are an extension of classical propositional logic with temporal operators. These logics are used to reason about *qualitative functional requirements* of real-time systems, which specify what a system should perform and what should not happen. The qualitative properties stated by these requirements are mainly *ordering of events* (event a is followed by event b), *liveness* (a request will be eventually responded), *safety* (event a will never occur), etc. These properties are extensively used in synchronous and reactive systems.

Below, we discuss the most influential propositional temporal logics, which have been extensively studied and widely used in various system specification and verification. Many extensions of these logics have been proposed, which will be discussed in the following sections.

3.1 Linear Time Logics

An important success in temporal logic study was the introduction of the temporal operators into the classical logic [46]. In [47] Pnueli introduced the influential *Linear Temporal Logic (LTL)*. LTL can express properties of linear sequences of states. For example, properties such as 'p holds at some state in the sequence' or 'p holds at two consecutive states in the future' can be expressed in LTL.

The LTL language consists of a finite set of *propositional variables* AP, the standard Boolean operators, i.e. $\neg, \wedge, \vee, \rightarrow$, and the *temporal operators* $\bigcirc$ ('next') and $\mathcal{U}$ ('until'). LTL formulas over AP can be defined as follows:

- $p \in AP$ is an LTL formula
- if φ_1 and φ_2 are LTL formulas, then $true, \neg\varphi_1, \varphi_1 \wedge \varphi_2, \bigcirc\varphi_1$ and $\varphi_1 \mathcal{U} \varphi_2$ are LTL formulas

We can derive other temporal operators, such as $\Diamond$ ('eventually') and $\Box$ ('always') from $\bigcirc$ and $\mathcal{U}$. For example, $\Diamond\psi \equiv true\ \mathcal{U}\ \varphi$ and $\Box \equiv \neg\Diamond\neg\psi$. Some example properties expressed in LTL are given below [48]:

- $p \rightarrow \Box q$: q holds at all states after p holds.
- $\Box((\neg q) \vee (\neg p))$: p and q cannot hold at the same time.
- $p \rightarrow \Diamond q$: q holds at some time after p holds.
- $\Box\Diamond p \rightarrow \Diamond q$: If p repeatedly holds, q holds after some time.
- $\Box p \rightarrow \Diamond q$: If p always holds, q holds after some time.

LTL formulas are generally interpreted over a *Kripke structure*, which is a tuple $\langle S, \rightarrow, L\rangle$, where S is a set of states, $\rightarrow$ is a transition relation and $L : S \mapsto 2^{AP}$ is a labeling function. When executed, a Kripke structure corresponds to a set of paths, which can be considered as a computation tree. Each path of the computation tree is an *infinite* and *linear* sequence of states, which is isomorphic to the linear temporal structure $\langle T, <\rangle$ defined in Section 2.

Let $\mathcal{M} = \langle S, \rightarrow, L\rangle$ be a model[4], σ be a path $s_0, s_1, ...$ in $\mathcal{M}$, where $s_i \rightarrow s_{i+1}$. The formal semantics of LTL formulas are defined as follows:

$$
\begin{aligned}
&\mathcal{M}, \sigma \models p \in AP \ \text{ iff } \ p \in L(\sigma[0]) \\
&\mathcal{M}, \sigma \models \neg\varphi \ \text{ iff } \ \mathcal{M}, \sigma \not\models \varphi \\
&\mathcal{M}, \sigma \models \varphi_1 \wedge \varphi_2 \ \text{ iff } \ \mathcal{M}, \sigma \models \varphi_1 \text{ and } \mathcal{M}, \sigma \models \varphi_2 \\
&\mathcal{M}, \sigma \models \bigcirc\varphi \ \text{ iff } \ \mathcal{M}, \sigma^1 \models \varphi \\
&\mathcal{M}, \sigma \models \varphi_1 \mathcal{U} \varphi_2 \ \text{ iff } \ \exists i \geq 0 \text{ s.t. } \mathcal{M}, \sigma^i \models \varphi_2 \text{ and} \\
&\qquad\qquad (\forall\, j < i)\ \mathcal{M}, \sigma^j \models \varphi_1
\end{aligned}
$$

where $\sigma[i]$ denotes the i^{th} element of the path σ and σ^i denotes the suffix $s_i, s_{i+1}, ...$ Informally speaking, the formula

[4] A *model* is an abstract representation of the behaviour of a system.

$\bigcirc\varphi$ holds at the current state if φ is true at the next state, and $\varphi_1 \mathcal{U} \varphi_2$ holds at the current state if φ_2 is true at some future state, and φ_1 is true at all moments until that future state. Basically, we say an LTL formula φ is true at a state $s \in S$ of the Kripke structure $\mathcal{M}$, if and only if, it is true at all paths σ starting s, which can be formally expressed as:

$$\mathcal{M}, s \models \varphi \text{ iff } \mathcal{M}, \sigma \models \varphi \text{ for all } \sigma \text{ starting } s.$$

In [49] LTL is defined over discrete time models with $\bigcirc$ and $\mathcal{U}$ operators. [49] shows that LTL is decidable, and provides a sound and complete axiomatization. In [50] Sistla and Clarke prove that the satisfiability and model checking problems of LTL are PSPACE-complete. [50] shows that if the syntax is restricted only to $\Diamond$ ('sometime') operator, or $\bigcirc$ operator, then the complexity of the satisfiability problem reduces to NP-complete. However, when both operators are included in the syntax PSPACE-completeness is preserved.

[50] provides a complete axiomatic system for LTL. Among the proof systems existing in the literature are a Hilbert-style proof system [14], a Gentzen-style proof system [48] and a clausal resolution approach [51, 52]. These proof systems are all sound and complete. In [53] LTL was extended with the past operators, and a complete proof system for both future and past operators was presented (A detailed discussion can be found in [48, 54]). It is known that adding past operators does not increase the expresiveness of LTL (when interpreted over left-bounded domains). In [54] an EXPTIME tableau algorithm is presented for the satisfiability problem of LTL.

Recently, more results have been presented on LTL. [55] shows that the satisfiability problem with the strict 'until' operator is PSPACE-complete. [56] extends the 'since-until' logic of real-line with the operators "sometime within n time units", and they show that the new logic is PSPACE-complete. [57] shows that satisfiability problem for the logic with 'since-until' operators over real-numbers time is PSPACE-complete.

3.2 Branching Time Logics

A temporal logic system is called a *branching time logic* if the underlying semantics of the structure of time is branching. The underlying structure of time in branching time logics is a tree-like structure. That is, every time instant can be followed by several immediate successor time instants. In branching time logics, there are two kinds of formulas: *state* formulas and *path* formulas. State formulas are interpreted over states and path formulas are interpreted over paths[5)].

Temporal logics with underlying branching time have found many applications in artificial intelligence. In particular, they are very useful in planning systems, where agents formulate different plans and action strategies according to different future world states (which have branching characteristics) [58, 59].

Since very efficient model checking algorithms have been introduced for branching time logics, these logics have been extensively used to verify finite state systems. On the other hand, in linear time logics deductive proof systems have been introduced for the verification of infinite state systems [3].

An initial work about branching time logics was done by [60]. Later, the unified branching time system (UB) was introduced in [61]. A simple branching time logic, CTL, was introduced in [62]. Thereafter, CTL* was introduced in [63]. CTL* is an extension over CTL by adding the properties of linear time temporal logic. CTL*[P], an extension over CTL*, was introduced in [64]. UB, CTL and CTL* include only future time temporal connectives, whereas CTL*[P] contains both past and future time temporal connectives.

3.2.1 Computational Tree Logic (CTL)

CTL is a point-based branching time logic, which is an extension of the logic UB by adding the operator $\mathcal{U}$. Time is included implicitly within the temporal operators, which allows us to express *some* or *all* computations. In CTL a discrete notion of time and only *future* modalities are used.

CTL formulas, φ, are defined according to the following grammar:

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists\psi \mid \forall\psi$$
$$\psi ::= \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where $p \in AP$, which is a set of atomic propositions. The formulas $\Diamond$ and $\Box$ can be obtained as in LTL.

CTL formulas are interpreted over *transition systems*. The execution of a transition system constructs a set of *paths*, which is an infinite *tree* of states, because the underlying time flow $\langle T, < \rangle$ is a *tree-like* structure and it satisfies the *branching* requirements discussed in Section 2. Assume $\mathcal{M} = \langle S, \rightarrow, L \rangle$ is a transition system (a Kripke structure). For a given CTL-formula φ and a state $s \in S$, the satisfaction relation $\models$ is inductively defined on the structure of φ as follows (here we skip standard logical formulas):

$$\mathcal{M}, s \models \exists\psi \text{ iff } \mathcal{M}, \sigma \models \psi \text{ for some } \sigma \in Paths(s)$$

5) Note that every state formula is also a path formula.

$\mathcal{M}, s \models \forall\psi$ iff $\mathcal{M}, \sigma \models \psi$ for all $\sigma \in Paths(s)$

Path quantifiers $\exists$ and $\forall$ represent *some path* and *all paths*, respectively, in the execution tree. The semantics of Path formulas is the following:

$\mathcal{M}, \sigma \models \bigcirc\varphi$ iff $\mathcal{M}, \sigma[1] \models \varphi$
$\mathcal{M}, \sigma \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists i \geq 0$ s.t. $\mathcal{M}, \sigma[i] \models \varphi_2$ and
$(\forall j < i)\ \mathcal{M}, \sigma[j] \models \varphi_1$

where $\sigma[i]$ denotes the i^{th} element of a path σ, and $Paths(s)$ denotes the set of paths from a state s.

Some example CTL formulas are given below [65]:

- $\exists\Diamond(p \wedge \neg q)$: There exists a state where p holds but q does not hold.
- $\forall\Box(p \rightarrow \forall\Diamond q)$: Whenever p holds, eventually q holds.
- $\forall\Box(\exists\Diamond p)$: At all paths p holds after some time.

CTL is a decidable logic [66]. It can be shown that CTL has the finite model property. That is, a satisfiable formula is satisfied in a finite model of size which is bounded by "some function of length of the formula" [67]. [68] presents a tableau method for checking the satisfiability of CTL formulas. The complexity of this procedure is EXPTIME. The model checking problem of CTL is easier than the satisfiability problem. Indeed, model checking in CTL is linear in the size of the model and the formula [69]. This shows that model checking in CTL can be achieved very efficiently. In [70] a sound and complete axiomatic system is provided for CTL.

3.2.2 CTL*

The logic CTL* was introduced in [63]. CTL* is an extension over CTL by adding the properties of linear time temporal logic. CTL* formulas, φ, are defined as follows:

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists\psi \mid \forall\psi$$
$$\psi ::= \varphi \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \bigcirc\psi \mid \psi_1 \mathcal{U} \psi_2$$

which suggests that the expressive powers of CTL and LTL are combined. As the above grammar suggests, the path quantifiers $\exists$ and $\forall$ can be arbitrarily nested with the operators $\bigcirc$ and $\mathcal{U}$. Since CTL* is more expressive than CTL, theoretical analyses become more difficult. Although model checking for CTL is linear, CTL* model checking is PSPACE-complete [69]. Also, solving the satisfiability problem for CTL* is more difficult than solving the CTL satisfiability. [71] provides an algorithm for the satisfiability problem of CTL*, which has 2-EXPTIME complexity in the length of the formula. A sound and complete axiomatisation for CTL* has recently been defined by Reynolds in [72].

3.2.3 CTL* with Past

In the logics CTL and CTL* we assumed that temporal operators are restricted to future time. [64] introduces a logic CTL*[P], which also includes past time operators. As in the linear case, addition of past operators to the language does not increase expressive power if we have a finite past; but this allows to express useful properties. Until recently the axiomatizability of CTL*[P] has been a long-lasting open question. [73] gives a sound and complete axiomatisation system for CTL*[P]. [74] proposes a translation algorithm to translateCTL*[P] formulas into CTL formulas to use CTL model checking, but no complexity result is provided. Recently, [75] has shown that the satisfiability problem of CTL* with linear past-time operators is 2-EXPTIME-complete, and the complexity of the model checking problem is between EXPTIME-complete and 2-EXPTIME-complete.

3.2.4 Expressiveness of Branching Temporal Logics

The expressive powers of CTL and LTL are not comparable. CTL allows quantification over paths, which LTL does not. For example, the CTL formula

$$\forall\Box\ \exists\Diamond\ p$$

asserting that 'it is always the case that p is eventually true', cannot be specified in LTL. On the other hand, the LTL formula

$$\Box\Diamond p \rightarrow \Diamond p$$

asserting that if 'p is infinitely true along the path, then q is eventually true', cannot be expressed in CTL. CTL* combines the expressive powers of CTL and LTL. For example, the CTL* formula

$$\exists(\Box\Diamond\ p)$$

asserting that 'there exists a path where p is infinitely true', cannot be expressed in either CTL or LTL.

UB and CTL can express important properties of concurrent programs, such as below [70]:

- $\forall\Box p$: *safety property*: p is true at all states of each path.
- $\forall\Diamond p$: *liveness property*: p is true at some state of each path.

Despite being able to express important properties such as safety and liveness, neither UB nor CTL can express fairness constraints. CTL* can be used for specification of more complex properties, which cannot be expressed by either UB and CTL. Some examples are given below [70]:

– $\Box\Diamond p \rightarrow \Box\Diamond q$: *fairness property*: if p is infinitely true, so is q.
– $\exists((pUq) \vee \Box p)$: *weak until property*

As seen above, the combination of branching and linear time operators result in more expressive power. This rich syntax enables to express more complex properties, such as fairness. The branching logics mentioned in this section can be made more expressive, while still keeping all their formulas as state formulas, by allowing classical operators between the temporal and path operators. If we add past operators, expressiveness does not increase; but the resulting logic allows more convenient notation to express some useful properties. Due to complexity and expressiveness considerations some other logics have been defined, such as CTL$^+$ [66], ECTL [68], ECTL$^+$ [68]. These logics are more expressive than CTL; but the complexities of the decidability and model checking problems are less than that of CTL*.

3.3 Partial-Order Temporal Logics

In concurrent systems computations are generally viewed as partially ordered sets. Since linear temporal logics are more suitable for totally ordered sets, it is difficult to apply them to concurrent and distributed systems [76]. Partial-order temporal logics are suitable to express partial orderings representing the behaviour of concurrent systems [77].

In branching time structures, each time instant may have several immediate successor points corresponding to different futures. Partial order structures are similar to branching structures except that each time instant can also have immediate predecessors corresponding to different pasts [70].

Initial attempts to define a logic based on partial orders were done in [77], where the logic POTL is introduced. POTL can express partially ordered computations without making any translation from totally orders sets. [77] shows that POTL does not have the finite model property due to the addition of past operators; but in spite of this negative result the authors show that the logic has an exponential decision procedure, and a complete axiomatisation system.

[78] introduces an extension of POTL, the logic POTL[U,S], which extends POTL with 'until' and 'since' operators. Similar to the case of POTL, POTL[U,S] can be seen as an extension of CTL with past modalities. POTL[U,S] can express all properties POTL can express as well as the properties concerning the "relative order of events in the future and past" [70]. [78] presents an exponential decision procedure. A sound and complete axiomatisation system for POTL[U,S] is also given in [78]. POTL[U,S] has a high model checking complexity. Indeed, [78] shows that the complexity is exponential in the model size and doubly exponential in the formulas size.

In the literature, there are more recent results for logics with partial-order semantics. [79] presents a new temporal logic, where linear and partial order semantics are combined. Namely, a computation is modeled as a linear sequence of states, which are associated with "past partial-order history". The authors also give a sound and partially complete proof system for the logic. In [80] partial order reductions are studied for the logics CTL and CTL* based on the partial order techniques to reduce the state space. [81] introduces a new partial-order temporal logic based on different semantical model to increase the expressiveness. In [82] partial-order reduction techniques are applied to linear and branching time temporal logics for knowledge (without the next operator) to reduce the model size before applying model checking procedure.

4 First-Order Temporal Logics

First-order temporal logics (FOTL) are extensions of propositional temporal logics. In addition to all propositional features these logics also allow arbitrary data structures and quantifiers. FOTLs have been extensively used in many areas including specification and verification of reactive systems, and analysis of hardware components. First-order logics provide an expressive formal framework for formalising the semantics of executable modal logics. They allow obtaining more robust techniques for reasoning about knowledge [83]. FOTLs have also found applications in information systems. For example, temporal database query languages are mainly based on first-order like languages [84].

Although first-order temporal logics have proved to be useful in various areas, they suffer from high computational complexity because these logics are very expressive. Indeed, most FOTLs are not even recursively enumerable [42,85,86]. Some axiomatisations of first-order temporal logics were studied in [87]. In some cases, fragments of first-order temporal logics with lower computational complexity are defined through restricted extensions to propositional temporal logics [85, 88–90].

One important logic is *monodic* fragment of first-order temporal logic, which is an expressive logic with a feasible computational behaviour. In monodic formulas, one free

variable is allowed at most in temporal subformulas [6]. In [91] a finite axiomatic system is presented for the monodic fragment. In [92, 93] monodic guarded decidable fragments are introduced by restricting the quantification.

Here, we consider FOTL as a representative of first-order temporal logics. FOTL has the following syntax (which does not comprise *equality* and *function symbols*) [94]:

predicate symbols: $P_0, P_1 ...$;
variables: $x_0, x_1, ...$;
constants: $c_0, c_1, ...$;
boolean connectives: $\vee, \neg$;
universal quantifier: $\forall$;
temporal operators: $\mathcal{U}$ ('until'), $\mathcal{S}$ ('since')

Let $\mathcal{M} = \langle \mathbb{T}, \mathcal{D}, \mathcal{I} \rangle$ be a first-order temporal model where $\mathbb{T} = \langle T, < \rangle$ is a strict linear order representing time flow, $\mathcal{D}$ is a non-empty domain set of $\mathcal{M}$, and $\mathcal{I}$ is a function assigning a first-order structure of the form

$$\mathcal{I}(t) = \left\langle \mathcal{D}, P_0^{\mathcal{I}(t)}, ..., c_0^{\mathcal{I}(t)}, ... \right\rangle$$

to every $t \in T$. For every i, $P_i^{\mathcal{I}(t)}$ is a predicate defined on $\mathcal{D}$ which has the same arity with P_i. The formal semantics of FOTL is defined as follows [94]:

$\mathcal{M}, \alpha, t \models P_i(x_1, ..., x_n)$ iff $P_i^{\mathcal{I}(t)}(\alpha(x_1), ..., \alpha(x_n))$ holds in $\mathcal{I}(t)$, where each x_i is eith.a variable or cons.
$\mathcal{M}, \alpha, t \models \neg\varphi$ iff not $\mathcal{M}, \alpha, t \models \varphi$
$\mathcal{M}, \alpha, t \models \varphi_1 \wedge \varphi_2$ iff $\mathcal{M}, \alpha, t \models \varphi_1$ and $\mathcal{M}, \alpha, t \models \varphi_2$
$\mathcal{M}, \alpha, t \models \forall x\varphi$ iff $\mathcal{M}, \beta, t \models \varphi$ for any β which differs from α at most in value of x
$\mathcal{M}, \alpha, t \models \varphi_1 \mathcal{S} \varphi_2$ iff $(\exists t' < t)$ $\mathcal{M}, \alpha, t' \models \varphi_2$ and $(\forall t'' : t' < t'' < t)$ $\mathcal{M}, \alpha, t'' \models \varphi_1$
$\mathcal{M}, \alpha, t \models \varphi_1 \mathcal{U} \varphi_2$ iff $(\exists t' > t)$ $\mathcal{M}, \alpha, t' \models \varphi_2$ and $(\forall t'' : t < t'' < t')$ $\mathcal{M}, \alpha, t'' \models \varphi_1$

where α and β are valuation functions which assign values from $\mathcal{D}$ to variables. The temporal operators $\bigcirc$, $\Diamond_F$ (sometime in future) and $\Box_F$ (always in future) can be derived from $\mathcal{U}$. Similarly, $\Diamond_P$ (sometime in past) and $\Box_P$ (always in past) can be derived from $\mathcal{S}$. Here, we give an example specification. The monodic formula

$$\forall x \Box_F(resp(x) \rightarrow \bigcirc \Box_F \neg resp(x))$$

informally means that the server responds each message only once. The following formula is not monodic, because the subformula $\bigcirc send(x, y)$ is with a temporal operator over a formula with two free variables:

$$\Box_P \Box_F(\neg(\exists y)\ send(x, y) \wedge \bigcirc send(x, y))$$

The above formula represents all processes that do not send the same message consecutively. Here, the variable x and y represents the process and message, respectively. Note that the syntax of the formulas above looks same in different domain sets, e.g. $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, etc. (based on finite or infinite domains), but their theoretical implications will be different.

Below, we assume FOTL($\mathbb{T}$) is the first-order temporal logic of $\mathbb{T}$, and FOTL$_{fin}(\mathbb{T})$ is the logic of $\mathbb{T}$ with *finite domains*.

4.1 Undecidable Fragments of FOTL

In the literature, it is known that both the monadic and two-variable fragments of first-order logic are decidable [95]. However, the computational complexities of their temporal counterparts are different. Let FOTL^2 denote the *two - variable fragment* of FOTL (where every formula contains at most two variables), and FOTL^{mo} denote the *monadic fragment* (not monodic) of FOTL (where formulas contain only unary predicates). Assume $\mathbb{T}$ is either $\{\langle \mathbb{N}, < \rangle\}$ or $\{\langle \mathbb{Z}, < \rangle\}$. Then, FOTL$^2 \cap$ FOTL$^{mo} \cap$ FOTL($\mathbb{T}$) and FOTL$^2 \cap$ FOTL$^{mo} \cap$ FOTL$_{fin}$ ($\mathbb{T}$) are not recursively enumerable [94].

4.2 Decidable Fragments of FOTL

The undecidable fragments given above have 3-dimensions in the presence of a temporal operator (introducing the temporal dimension) over a formula with two free-vars (introducing the 2nd and 3rd dimension). This is the common property of the undecidable fragments of FOTL, and it is the cause of bad computational behaviour. It is known that the two-variable fragment of first-order logic which are not monodic is undecidable [95].

In order to preserve decidability, the monodic fragment, denoted as FOTL$_1$, has been introduced. FOTL$_1$ contains FOTL formulas φ such that any sub-formulas of φ of the form $\varphi_1 \mathcal{U} \varphi_2$ and $\varphi_1 \mathcal{S} \varphi_2$ has at most one free variable. The monodic fragments of FOTL($\langle \mathbb{N}, < \rangle$) and FOTL($\langle \mathbb{Z}, < \rangle$) are recursively enumerable [96].

Let $\mathbb{T}'$ be $\{\langle \mathbb{R}, < \rangle\}$ and $\mathbb{T}$ be the following classes of time structures: "$\{\langle \mathbb{N}, < \rangle\}$, $\{\langle \mathbb{Z}, < \rangle\}$, $\{\langle \mathbb{Q}, < \rangle\}$, the class of all finite strict linear orders, and any first-order-definable class of strict linear orders" [96]. [94] proves that various fragments are decidable, such as FOTL($\mathbb{T}$) $\cap$ FOTL$_1$, FOTL($\mathbb{T}$) $\cap$ FOTL$_1^2$, FOTL($\mathbb{T}$) $\cap$ FOTL$_1^{mo}$, FOTL$_{fin}(\mathbb{T}') \cap$ FOTL^1, FOTL$_{fin}(\mathbb{T}') \cap$

[6] A formula is said to be monodic if it has no subformulas of the kind $\varphi \mathcal{U} \psi$ or $\varphi \mathcal{S} \psi$ with more than one free variable.

FOTL_1^2 and $\text{FOTL}_{fin}(\mathbb{T}') \cap \text{FOTL}_1^{mo}$. They also provide some *guarded* fragment of first-order language (For a detailed discussion, see [94]).

In [97] it is shown that $\text{FOTL}(\langle\mathbb{N}, <\rangle) \cap \text{FOTL}_1$ is EXPSPACE-hard. It is also shown that "the satisfiability problem for FOTL_1^{mo}-formulas in models based on $\langle\mathbb{N}, <\rangle$ is EXPSPACE-complete" [96].

Here we assumed that FOTL and its fragments do not include *equality* and *function symbols*. It can be shown that undecidability is a major problem with the logic extended with function symbols [91]. For instance, "the set of one-variable formulas with one function symbol that are valid in models based on $\langle\mathbb{N}, <\rangle$ is not recursively enumerable" [96]. Moreover, "the set of monodic FOTL formulas with equality that are valid in all temporal models based on $\langle\mathbb{N}, <\rangle$ is not recursively enumerable" [96]. [98] shows that the problem persists even for a simpler fragment. Namely, the authors prove that a fragment with "monodic monadic two-variable formulas" is not recursively enumerable. In [91] a finite *Hilbert-style* axiomatisation of the monodic fragment of first-order temporal logic was presented. It was also proved that "the monodic fragment with equality is not recursively axiomatisable" [91].

Recent research results have showed that relatively expressive subsets of first-order temporal logic could be found. [91, 94, 99] suggest that the expressive power of monodic first-order temporal logic can be extended further. The decidability results can be also be extended to temporal description logics. Recently, tableau-based methods are presented for the satisfiability checking of temporal description logics [100]. Tableau-based methods can also be devised for the satisfiability checking of decidable monodic temporal logics. This can be done by extending the tableau methods for the propositional temporal logics to the first-order case [100]. An alternative approach is to use the resolution method. [101] introduces some resolution systems for monodic first-order temporal logics.

5 Real-Time Temporal Logics

Real-time temporal logics have been introduced to formally specify and reason about *qualitative* and *quantitative functional requirements* of real-time systems, which specify what a system should perform and what should not happen. The qualitative requirements have already been discussed in Section 3. The *quantitative* requirements express the properties about "time-critical properties that relate the occurrence of events" [102]. Some typical quantitative properties are as follows [103]:

- *distance between two states*: an event A is followed by an event B within minimum/maximum/exactly 5 seconds.
- *bounded response time*: there is an upper time limit for a respond's to arrive after the occurrence of an event A.
- *periodicity*: an event A occurs repeatedly every 5 seconds.

Below we give a brief account of well-known real-time temporal logics. All these logics are different in terms of 'expressiveness', 'order', 'time metric', 'temporal modalities', 'time model' and 'time structure'. They also have different capabilities for the specification and verification of real-time systems.

Note that the majority of the logics discussed in this section are point-based. We include some interval logics (e.g. RTIL, TILCO and TRIO) in this section because, they are closely related.

5.1 Real-time Extensions of LTL

There have been numerous attempts to extend LTL with quantitative operators to capture quantitative (metric) real-time properties, which cannot be expressed in classical LTL. Due to the abundance of these logics, we will only highlight important ones. The reader can consult to the referred sources for a more detailed account.

The most widely studied and analysed real-time extension of LTL is MTL [103], where time references are added to temporal operators ('until', 'next' and 'since'). Due to its importance among other extensions, MTL will be discussed in more detail in the next section.

TPTL [104] is another real-time extension of LTL, interpreted over infinite discrete sequences of states. TPTL allows expressions with arithmetic operations; but this is only allowed for integer constants (not for variables). In TPTL explicit reference to clock is replaced by *freezing* quantification, and clock values are recorded through "auxiliary static timing variables" [3]. Namely, TPTL extends the language of LTL with clock constraints and a freezing quantification. This feature allows us to write real-time properties, which refers to the exact system clock. For example, in the following formula,

$$\Box x.(p \rightarrow (x \leq 5))$$

asserting that whenever p is true it is satisfied in a state when the clock value is less than 5, where $x \leq 5$ is a clock constraint and $x.(..)$ is a freezing quantification. Another real-time requirement we can specify with TPTL is that the system must

switch to the state q within 5 seconds after being in the state p, which is expressed in TPTL as follows [104]:

$$\Box x_1.(p \rightarrow p\ \mathcal{U}\ x_2.(q \wedge x_2 \leq x_1 + 5))$$

The satisfiability and model checking problems for TPTL with discrete time semantics are EXPSPACE-complete; but they become undecidable with dense time semantics [104]. [104] also presents a doubly-exponential-time decision procedure for TPTL. The model checking algorithm for the logic is "exponential on the value of the product of all time constants" [3]. [2] shows that if past operators are added to the logic, the satisfiability problem for TPTL becomes non-elementary. [105] proves that there is a complete finite axiomatization for TPTL with discrete time semantics.

[106] introduces the logic RTPLTL, which is an extension of LTL employing a "tractable fragment of regular expressions" and quantitative constraints on the number of the states where sub-expressions occur. RTPLTL can express regular sequences over paths (as well as properties expressible in LTL). This is shown in the following example. Assume that $\overline{str1}^{*}\ str2$ denotes a string of system actions where $str1$ does not occur in the string and $str2$ is the last element, and $(\overline{str1}^{*}\ str2)^n$ denotes n successive occurrences of $\overline{str1}^{*}\ str2$; i.e. $str2$ occurs n consecutive times without any occurrence of $str1$. The statement 'if the server has received three consecutive requests and it has not responded yet, then the server will respond before the fourth request arrives' is expressed in RTPLTL as follows [106]:

$$\Box\left((\overline{req + resp}^{*}\ req)^3 true \rightarrow \left((\overline{resp}^{*}\ resp) \cap (\overline{req}^{*}\ req)^{\leq 3}\right) true\right).$$

RTPLTL employs an automata-theoretic model checking algorithm whose time complexity is exponential in the size of formulas. The main difference between TPTL and RTPLTL is that RTPLTL is not restricted to models involving a single time sequence.

Another quantitative extension, called CLTL, was introduced in [107], where LTL is extended with *counting* quantification on the number of states where a certain subformula is true. Namely, the standard temporal $\mathcal{U}$ modality is replaced with $\mathcal{U}_{[C]}$ which employs the counting quantification C. For example, the statement 'p eventually becomes true, and until then q holds at most 3 times' is expressed in CLTL as

$$\Box\left(\Diamond_{[\sharp q \leq 3]}\ p\right).$$

where $\sharp q$ represents the number of states where q is true. We can easily adapt this type of specifications to specify important real-time properties, such mutual exclusion protocol, where two processes try to enter the critical section. For example, the following specification asserts that whenever the process 1 sends a request to enter the critical section, it will do so after the process 2 enters the critical section at least 3 times [107]:

$$\Box\left(req_1 \rightarrow \Diamond_{[\sharp cs_2 \geq 3]}\ cs_1\right).$$

CLTL formulas are interpreted over infinite discrete words; that is the logic employs discrete linear time flow. [107] shows that satisfiability and model checking problems of this extension are both EXPSPACE-complete, and translating a formula of this extension to the equivalent LTL formula leads to an exponential blow-up on the formula size. They also show that if the constraints are extended with subtraction, then both satisfiability and model checking problems become undecidable. CLTL can express more complex quantitative constraints, but cannot express ordering of events, which RTPLTL can express.

5.2 Metric Temporal Logic (MTL)

MTL [103] is a propositional bounded-operator logic, which is an extension of LTL with timing constraints. Namely, time references are added to temporal operators. This is done by replacing the $\mathcal{U}$ operator of LTL with $\mathcal{U}_I$, where I is an interval of reals with endpoints in $\mathbb{N} \cup \infty$. In MTL explicit reference to clock is not allowed, which makes the logic more practical because quantifications on a temporal domain are no longer needed. MTL can express deadline properties, i.e. when an action occurs, the system must respond in a certain period. For example, the formula

$$p \rightarrow \Diamond_{(0,10)} q$$

asserts that if p occurs then q occurs within 10 time units. More specifically, the property "every alarm is followed by a shutdown event in 10 seconds unless all clear is sounded first" is expressed in MTL as follows [108]:

$$\Box(alarm \rightarrow (\Diamond_{(0,10)} allclear\ \vee\ \Diamond_{\{10\}} shutdown))$$

where $(0, 10)$ states 'within 10 seconds' and $\{10\}$ states in exactly 10 seconds. MTL is interpreted over linearly ordered time domains, which can be either discrete, dense or continuous. The semantics varies depending on the choice of the time flow. Assume that $f : \mathbb{R}_{>0} \rightarrow 2^P$ is a mapping from a real-time point t to the set of propositions holding at time t, where f has the *finite variability* property, i.e. the number of discontinuous points of any state in any unit interval has a fixed upper bound. The formal semantics based on dense

time is given as follows [108]:

$f \models \varphi_1 \mathcal{U}_I \varphi_2$ iff $\exists t \in I$ s.t. $f^t \models \varphi_2$ and $(\forall t' \in (0,t))\ f^{t'} \models \varphi_1$.

where $f^t(s) = f(t+s)$. The semantics based on *timed words* can be formulated as follows [108]:

$$\sigma[i] \models \varphi_1 \mathcal{U}_I \varphi_2 \text{ iff } \exists j \geq i \text{ s.t. } \sigma[j] \models \varphi_2, (t_j - t_i) \in I \text{ and } (\forall i \leq k < j)\ \sigma[k] \models \varphi_1$$

where a timed word σ is a finite or infinite sequence $(e_0, t_0), (e_0, t_1), ...$, where $t_i \in \mathbb{R}_{>0}$ and $e_i \in E$ (E is a set of events).

In [103], dense time domain is considered. This allows MTL to express properties which cannot be precisely expressed in a discrete-time domain, such as variables based on continuous time (e.g temperature and pressure). [2] states that both the satisfiability and model checking problems for MTL over dense time domain are undecidable, but a deductive proof system exists. [103] provides a sound axiomatic system for MTL.

Because of the undecidability results of MTL, researchers have considered various ways to obtain decidability. One approach is to restrict time domain: [2] shows that in case of discrete time both satisfiability and model checking problems reduce to EXPSPACE-complete. [2] also introduces a decision procedure for MTL over discrete time domain, which has 2-EXPTIME complexity, and a model checking algorithm, which is exponential on the value of the largest time constant. [109] finds that the satisfiability problem for MTL over finite timed words is decidable, but it has non-primitive recursive complexity. If we assume infinite words, both satisfiability and model checking problems become undecidable.

Another option to achieve decidability is to restrict the syntax: $\text{MTL}_{0,\infty}$ [110] is a subset of MTL, which is obtained by restricting the interval I such that either the start point of I is 0, or the end point of I is ∞. This can be considered as adding upper and lower bound timing constraints to the operator $\mathcal{U}$. For example, the following $\text{MTL}_{0,\infty}$ formula

$$\Box(p \rightarrow \Diamond_{(10,\infty)} q)$$

states that p must be followed by q in more than 10 seconds. The satisfiability and model checking problems of $\text{MTL}_{0,\infty}$ are both PSPACE-complete [110].

In [110] MTL is restricted to "interval-based strictly-monotonic real-time semantics". This logic is called MITL, which uses operators with a bound. The authors state a close relationship between undecidability and punctuality properties[7] over dense time. For this reason, point intervals are not allowed in MITL, and thus it cannot express punctuality properties. For example, the formula

$$\Box(p \rightarrow \Diamond_{\{3\}} q)$$

is not a valid formula because equality constraints are not allowed. [110] shows that the satisfiability and model checking problems for MITL were shown to be EXPSPACE-complete. Regarding the expressivity, [111] shows that MTIL and $\text{MTL}_{0,\infty}$ have the same expressive power under dense semantics; and [112] shows that MTIL is strictly more expressive than $\text{MTL}_{0,\infty}$ under discrete semantics.

Recently, [109, 113] have shown that restricting MTL to positive-length intervals is not necessary to achieve the decidability. They show that "MTL over finitary event-based semantics" are decidable without this restriction. [114] compares the past and future fragments of MITL with respect to the "recognizability of their models by deterministic timed automata". The authors show that "timed languages specified by the past fragment of MITL, can be accepted by deterministic timed automata; but certain languages expressed in the future fragment of MITL are not deterministic."

[5] introduces the logic QTL, which is a variant of MITL. QTL replaces $\mathcal{U}$ operator of LTL with $\mathcal{U}_I$, where I is an interval of reals with endpoints in $(0, t)$, where $t \in \mathbb{N}$. In addition to future 'until' $\mathcal{U}$ operator, QTL also includes past time 'since' operator. [5] shows that the satisfiability problem is in PSPACE. [5] also shows that QTL has the same expressive power as MITL with 'past' operator.

[115] obtains a decidable sub-logic of MTL, called BMTL, where all constraining intervals have finite length. For example, the formula

$$\Box_{(0,5)}(p \rightarrow \Diamond_{(0,10)} q)$$

is a BMTL formula; but

$$\Box(p \rightarrow \Diamond_{(0,10)} q)$$

is not a BMTL formula, because $\Box$ is not constrained. Both satisfiability and model checking problems of BMTL are EXPSPACE-complete. MITL and BMTL have different strengths, and the expressive powers of these two logics are not comparable.

[115] also defines another subset of MTL, called CFMTL, which subsumes both MITL and BMTL. Namely, CFMTL includes a dual until operator $\widetilde{\mathcal{U}}_I$ as well as $\mathcal{U}_I$, and formulas satisfy the following conditions:

$\varphi_1\ \mathcal{U}_I\ \varphi_2$: either I has a finite length or φ_2 is MITL formula
$\varphi_1\ \widetilde{\mathcal{U}}_I\ \varphi_2$: either I has a finite length or φ_1 is MITL formulawhere $\varphi_1\ \widetilde{\mathcal{U}}_I\ \varphi_2 \equiv \neg(\neg\varphi_1\ \mathcal{U}_I\ \neg\varphi_2)$. For example, the

[7] A *punctuality* property states that the event B follows A in exactly t seconds.

following formula

$$\Box(p \rightarrow (\Diamond_{(0,10)} q \ \vee \ \Diamond_{\{10\}} r))$$

is in CFMTL; but not in MITL because of the punctuality operator {10} and not in BMTL because of the unbounded operator □. The satisfiability problem for CFMTL is undecidable; but the model checking problem is EXPSPACE-complete.

We finally report some comparison results. [116] has proven that the logic TPTL is strictly more expressive than MTL. This can be shown through the following real-time specification: "within 5 time units after the occurrence of a problem, the system triggers the alarm and then enters a fail-safe mode", which cannot be expressed in MTL, but can be expressed in TPTL as follows [116]:

$$\Box x.(problem \rightarrow x.\Diamond(alarm \ \wedge \ \Diamond \ (failsafe \ \wedge \ x \leq 5))).$$

5.3 Real-time Extensions of CTL

In [102] a real-time extension of CTL, called RTCTL, was introduced. RTCTL has "point-based strictly-monotonic integer-time semantics" [117]. The logic includes a metric for time, it can therefore express quantitative temporal assertions, such as "time-critical correctness properties of programs". RTCTL replaces the until operator $\mathcal{U}$ of CTL with the 'bounded-until' $\mathcal{U}^{\leq k}$ operator, where k is a constant, the semantics of which is defined as follows:

$$\mathcal{M}, \sigma \models \varphi_1 \mathcal{U}^{\leq k} \varphi_2 \ \text{ iff } \ (\exists i) 0 \leq i \leq k \text{ s.t. } \mathcal{M}, \sigma[i] \models \varphi_2 \text{ and } (\forall j < i) \ \mathcal{M}, \sigma[j] \models \varphi_1$$

Intuitively, $\varphi_1 \mathcal{U}^{\leq k} \varphi_2$ holds at the current state if φ_2 is true within k steps, and φ_1 is true until then. For example, the following formula

$$\forall(p \ \mathcal{U}^{\leq 5} \ q)$$

asserts that p must be followed by q within 5 seconds. As in CTL, the underlying time is discrete and RTCTL formulas are interpreted over discrete transition systems.

The satisfiability problem of RTCTL is EXPTIME-complete without 'equality'[8] and 2-EXPTIME-complete with 'equality' [102]. The model-checking problem is linear in the sizes of both formula and model [102].

[118] introduced the logic TCTL, which is a real-time extension of CTL, where constraints on duration are added to temporal operators. Similar to the logic RTCTL, TCTL replaces the CTL operator $\mathcal{U}$ with $\mathcal{U}^{\sim k}$, where $\sim \in \{<, >, \leq, \geq, =\}$ and $k \in \mathbb{N}$. As can be seen, TCTL employs a richer set of relational operators than RTCTL. The operators $\Diamond^{\sim c}$ and $\Box^{\sim c}$ can be derived in usual way. These operators are extensions of the standard $\Diamond$ and $\Box$ operators with timing constraints. For example, the formula $\exists\Diamond^{\leq 5} on$ states that "the system will be in the 'on' state within 5 time units".

Unlike RTCTL, TCTL employs a dense-time, and its formulas are interpreted over dense time structures, which have an infinite sequence of states. A path σ of a dense-time structure is defined as an infinite sequence $s_0 \xrightarrow{\xi_0} s_1 \xrightarrow{\xi_1} s_2 \xrightarrow{\xi_2} \ldots$, where $\xi_i \in \mathbb{R}_+$ (for $i \in \mathbb{N}$) denotes time passage. As defined before, the i^{th} element of σ is denoted by $\sigma[i]$. The semantics of $\mathcal{U}^{\sim k}$ operator can then be defined as follows:

$\mathcal{M}, \sigma \models \varphi_1 \mathcal{U}^{\sim k} \varphi_2$ iff
$\exists i \geq 0$ s.t. $\mathcal{M}, \sigma[i] \models \varphi_2, \mathcal{T}(\sigma[0] \overset{\sigma}{\mapsto} \sigma[i]) \sim k$, and
$(\forall j < i) \ \mathcal{M}, \sigma[j]) \models \varphi_1$

where $\mathcal{T}(\sigma[0] \overset{\sigma}{\mapsto} \sigma[i])$ denotes the elapsed time of reaching a state $\sigma[i]$ from a state $\sigma[0]$ within the path σ, which is the sum of the delays along the path. Intuitively, the formula $\varphi_1 \mathcal{U}^{\sim k} \varphi_2$ holds if φ_2 is true at some time point in a path, the elapsed time of reaching this point within the path satisfies $\sim k$, and at all previous time points φ_1 is true. Here we give an example to show how the operators are nested. The formula

$$\forall\Box(p \rightarrow \forall\Diamond^{\geq 5} q)$$

states that after staying in the $p-$state the system must switch to the $q-$state within 5 seconds.

Due to its dense time semantics, TCTL can model phenomena which require continuous-time in nature. Although discrete-time semantics is sufficient for synchronous systems, dense-time models asynchronous systems (e.g. distributed systems) more accurately, because these systems employ components which may have different clock cycles. Therefore, it is more adequate to use dense time (or continous time) than discrete time.

The satisfiability checking of a TCTL formula is undecidable if it is interpreted over dense time domains; but the model checking problem still remains decidable [118], which finds that the model checking complexity of TCTL is "exponential in the number of clocks, exponential in the length of timing constraints, linear in the size of the node-transition graph, linear in the number of operators in the formula and exponential in the length of the subscripts in the formula". [118] also shows that the upper bound can be improved to PSPACE, and the model checking problem is PSPACE-complete. [119] considers the model checking problem of different subclasses of TCTL.

[8] That is, the temporal operators of the form $\mathcal{U}^{=k}$ are not allowed.

Another branching time logic called TPCTL is introduced in [120]. TPCTL is a probabilistic and real-time extension of CTL, where the 'until' $\mathcal{U}$ modality of CTL is replaced by $\mathcal{U}^{\lhd k}_{\trianglerighteq p}$, where $\trianglerighteq \in \{>, \geq\}$, $k \in \mathbb{N}$ and $0 \leq p \leq 1$. $\mathcal{U}^{\lhd k}_{\trianglerighteq p}$ informally means that 'within k time units with a probability at least p'. TPCTL can express both hard and soft deadline properties, such as 'an error occurs with a probability at least 0.5 within 10 seconds', which is expressed in TPCTL as follows:

$$\exists(true\ \mathcal{U}^{\leq 10}_{\geq 0.5}\ err).$$

TPCTL semantics is defined over non-deterministic probabilistic transition systems, and the underlying time structure is represented by discrete time. TPCTL can also be regarded as a probabilistic extension RTCTL. The main difference is that they are interpreted over different structures. TPCTL is a decidable logic, and the model checking problem was proven to be polynomial [121].

Recently, another quantitative extension of CTL, called CCTL, was introduced in [122], where CTL is extended with counting quantification (denoted by $\sharp$) and formulas are interpreted over infinite discrete words. For example, the statement 'p eventually becomes true, and until then q holds at most 3 times' is formally expressed as follows:

$$\forall\Box\ \exists\Diamond_{[\sharp q \leq 3]}\ p$$

where p and q are atomic propositions, and $\sharp q \leq 3$ is a constraint over the number of states (here $\sharp q$ represents the number states where q is true).

[122] provides an analysis of the expressiveness and the complexity of the model-checking problem for a range of quantitative extensions. Depending on the extension, different complexity results are obtained. If temporal operators are restricted to *atomic* constraints (which has the form $\sum_i \sharp\psi_i \sim c$) or *diagonal* constraints (which has the form $\sum_i \mp\sharp\psi_i \sim c$), where $c \in \mathbb{N}$ and $\sim \in \{<, >, =, \leq, \geq\}$, then a polynomial model checking complexity is obtained. If the Boolean combinations of atomic constraints are allowed, the model checking becomes NP-complete. If the Boolean combinations of diagonal constraints are allowed, the model checking becomes undecidable. If the *freeze* variables are used instead of counting constraints, then the model checking problem becomes PSPACE-complete.

5.4 Real-Time Logic (RTL)

RTL, introduced in [123], extends first-order temporal logic with a set of constructs to reason about events and their relations. It provides a suitable syntax to specify relative and absolute timing of events. The logic includes a so-called *occurrence* function which maps each event to a time stamp. The existence of an occurrence function allows RTL to express periodic and non-periodic real-time properties. In RTL, time is measured with an 'absolute' clock whose value can be referenced in a formula.

RTL extends integer arithmetic with operators to reason about occurrences and relations of events. An action A is represented by two events: $\uparrow A$, denoting the start of the action A, and $\downarrow A$, denoting the finish of the action A. Any external event B is prefixed by a special symbol ΩB. The occurrence function @ assigns time values to event occurrences. Namely, $@(a, i)$ denotes the time of the ith occurrence of the event a.

RTL formulas are formed by *state predicates, path quantifiers* $(\exists, \forall)$ and *first-order logic connectives*, where a state predicate asserts the truth value of a state attribute during an interval, and it is constructed from *constants, variables (events, actions), relational operators* $(<, >, =, \leq, \geq)$, *arithmetic operators* and the *occurrence function* [124]. Since absolute clocks are used, and clock values can be explicitly referenced in formulas, RTL can be used to express ordering and quantitative temporal constraints. One disadvantage of this functionality is that using explicit reference to time results in complex formulas difficult to understand. For example, the following RTL formula

$$\forall i\,[@(\Omega B, i) < @(\uparrow A, i) \wedge @(\downarrow A, i) \leq @(\Omega B, i) + t_b]$$

states that the action A is executed just after the external event B occurs, and every execution of the action A is finished within t_b seconds after B occurs [124].

RTL is defined over a linear sequence of discrete time points, which are bounded in the past, but unbounded in the future. [2] shows that under these semantics RTL is undecidable. Some decidable fragments of have been defined, such as 'path RTL' [125] and 'extended path RTL' [126], which can only express timing constraints between two events. Recently, a more expressive decidable fragment, called LRTL, has been introduced. LRTL subsumes both 'path RTL' and 'extended path RTL' and it can express timing constraints of more than two events; but it cannot specify 'arithmetic expressions with a function may take an instance of itself as an argument' [127].

Early model checking procedures devised for RTL in general are not practical. To increase the efficiency some methods were deployed. In [125], RTL formulas are re-structured into "computational graphs" using a formalism called "modecharts", which resulted in "an exponential time decision pro-

cedure (in the worst case)". [126] uses decomposition techniques (decomposing the constraint graph to subgraphs) to improve model checking. [127] improves efficiency by using linear algebra techniques rather than computational graphs. This method has polynomial-time complexity.

5.5 Real-Time Temporal Logic (RTTL)

RTTL [128, 129] is a first-order explicit clock logic. Discrete linear time points are employed as temporal structure. The sequence of time points are bounded in the past, but unlimited in the future. In an RTTL formula the clock variable t is explicitly referred. As an example, "the bounded response time" is expressed in RTTL as follows [3]:

$$\Box T[(red \wedge t = T) \rightarrow \Diamond(green \wedge T + 3 \leq t \leq T + 5)]$$

which means that "if the traffic light is *red* at time T, then eventually within 3 to 5 ticks from T the light must turn *green*". Above t is the clock variable, and T is time variable, which is quantified in the formula.

RTTL provides an explicit reference to clock value and indirect quantification to time values. This results in a very expressive language, and allows writing very complex quantitative constraints. This makes this logic very useful in real-time system specification. However, undecidability is a major problem. In addition, due to explicit clock reference, formulas become too complex and difficult to understand.

In addition to discrete semantics, RTTL formulas can be also interpreted over a dense time domain. The logic is undecidable in both discrete and dense semantics [2]. The model checking in RTTL is also undecidable. RTTL has a sound proof system [129].

Although RTL is not a more expressive logic than RTTL, it can express certain properties more succinctly than RTTL; but RTL cannot easily represent fairness properties and data variables [129]. Some decidable fragments of RTTL are presented in the literature. Some well-known fragments are as follows: XCTL [130] is a propositional fragment of RTTL. It is an explicit clock logic, and it is interpreted over discrete time. XCTL has a less restricted quantification than RTTL in the sense that time variables can be quantified with only one outermost quantification; but the syntax of XCTL allows expressions with arithmetic operations. In [131] it is shown that XCTL and MTL cannot be compared; namely, for both logics, there is a property which is expressible in one logic, but not in the other [3]. The satisfiability and model checking problems for XCTL with dense time semantics are both undecidable [130]. However, these problems are PSPACE-complete for XCTL without quantification [130]. [130] provides a "single exponent decision procedure for the validity of XCTL formulas" and a "double exponent procedure" for XCTL model checking.

5.6 Real-Time Interval Logic (RTIL)

RTIL [16] is a propositional real-time interval logic with metric for time. It can express event properties to reason about events and their relations. It integrates both 'time-stamps' and 'realtime', and therefore it can specify properties like 'the value of a certain variable is 1 between the time value 2 and 5'. Note that this property cannot be specified by RTL and RTTL unless these time points coincide with an occurrence of an event.

RTIL allows assigning numerical values to interval bounds and to measure interval durations. It also allows quantification over finite domains. Intervals are sequences of discrete points. Time points can be specified explicitly or relative to the beginning of the interval [1]. These characteristics make RTIL to be useful in formalise specifications in a neater syntax. For example, the temporal constraint "for each occurrence of an event B which happens at a time instant t_0, the propositions *startA* and *endA* hold (marking an interval $[startA, endA]$ at which A is true), and the interval $[startA, endA]$ is subsumed by the interval $[t_0, t_0 + t_b]$ (where $t_0 \leq startA \leq endA \leq t_0 + t_b$)" is specified in RTIL as follows [1]:

$$\Box\,[\odot B \hookrightarrow t_b]^{*}\,(\odot startA \rightarrow \odot endA)$$

where $\odot A$ extracts the time point at which A becomes true, and the operator $*$ means there exists a subinterval. RTIL is a highly undecidable logic.

5.7 Tempo Reale ImplicitO (TRIO):

TRIO [132] is an extension of first-order logic with metric operators, which allows expressing quantitative real-time properties such as distance between two events and length of an interval. It is interpreted over linearly and totally ordered time domains, which can have a variety forms, such as dense, finite discrete, infinite discrete etc.

In addition to standard FOL operators and quantifications, the TRIO alphabet includes two temporal operators associated with a time metric: Futr(A, t) and Past(A, t), which denote that A occurs at the time instant t in the future and past, respectively. The temporal operators 'since', 'until', 'sometime' and 'always' can be derived from these two operators.

TRIO allows quantification on temporal variables and allows expressing the ordering (relation) between temporal variables [1]. For this reason, TRIO formulas are, in general, complex and the readability is hard. The specification example in the previous section can be written in TRIO as follows [1]:

$$\begin{aligned}\text{Alw}(B \rightarrow \exists t((0 < t < t_b) \wedge \text{Futr}(endA, t) \wedge \\ \exists t'(0 < t' < t \wedge \text{Futr}(startA, t')))\end{aligned}$$

where $\text{Futr}(A, t)$ denotes that A occurs at a time instant t in the future, and Alw states that A holds in every time instant of the temporal domain, which is formally defined as

$$\forall t(t > 0 \rightarrow \text{Futr}(A, t)) \wedge A \wedge \forall t(t > 0 \rightarrow \text{Past}(A, t)).$$

Since TRIO is an extension of first-order logic, it is undecidable. Satisfiability can be achieved if a restriction is put on variable valuation domains and temporal domains. Namely, if the temporal domain is restricted to limited intervals of integer values, and every variable valuation domain is assumed finite, then the satisfiability problem of TRIO becomes decidable [132].

Since TRIO does not have an axiomatic system, deductive verification techniques cannot be applied. However, a specific model checking method can be defined for this logic. TRIO can be considered as an executable logic. Therefore, the model of a system can be constructed using TRIO formulas. Then, "histories of system variables" can be checked against a formal specification [1].

5.8 Temporal Interval Logic with Compositional Operators (TILCO):

TILCO [133, 134] is an extension of first-order logic with temporal operators. It is an interval logic; that is, the logic is interpreted over linear intervals. TILCO does not provide explicit temporal quantification on time points; but operators quantify over intervals.

TILCO provides metric for time and can specify qualitative and quantitative timing constraints. Namely, end points of an interval at which an action or an event holds can be specified with respect to that of other events of actions; in addition, this can be done with an absolute numerical measure. This makes TILCO a very expressive logic, and very useful to specify complex behaviours of real-time systems.

In TILCO, primitive temporal objects are intervals, which are linear and composed of discrete time points. Intervals are constructed by the delimiters $(,), [,]$. In addition to the standard operators, TILCO includes the following temporal operators:

the *universal quantification* @ over intervals, ? *existential quantification* over intervals, Until and Since.

$\varphi@[t, t']$ is true if the formula φ is true at every point in the interval $[t, t']$, and $\varphi?[t, t']$ is true if the formula φ is true at some point in the interval $[t, t']$..

Since TILCO is an interval-based logic, it is more natural to specify temporal constraints with time bounds. Therefore, TILCO is very efficient to express "invariants, precedence among events, periodicity, liveness and safety conditions, etc." [134]. The specification in the previous section, i.e. the action A is executed just after the external event B occurs, and every execution of the action A is finished within t_b seconds after B occurs, can be expressed in TILCO as follows [1]:

$$B \rightarrow endA?(0, t_b) \wedge \neg\text{Until}(endA, \neg startA)$$

The logic is unsurprisingly undecidable, because it extends first-order logic. However, a decidable subset can be obtained if the variables with finite domain are binded by nontemporal quantification. The complexity of validity checking for this subset is exponential in the worst-case [134]. [133, 135] provide a sound sound deductive system. This proof system is used along with the Isabelle theorem prover [136] to provide an automatic proof tool for TILCO. [137] devices a tool which executes TILCO specifications and generate programming language source codes automatically. A property to be executed is assumed to be a verified and validated specification.

In [138], TILCO has been extended with new syntax and semantics of quantifications @ and ?, where dynamic intervals are used, counting of events over intervals are possible, and Until and Since operators are replaced with a simpler syntax [139]. [140] also extends TILCO, which supports process composition and decomposition by allowing the specification of communicating TILCO processes [139].

6 Interval Temporal Logics

Interval temporal logics are temporal logics which allow reasoning about periods of time. Since representation formalisms based on intervals are more expressive than formalisms based on time points, the interval-based scheme provides us with a richer representation formalism than the point-based approach. Intervals logics can also be used in reasoning about functional requirements of real-time systems.

In this section, we present a selection of well-known interval temporal logics. In the literature, many similar logics can be found; but most of these logics are generalisations or specialisations of the ones we will discuss below.

6.1 Propositional Interval Temporal Logics

In this section we will present the well-known propositional interval logics, which involve unary or binary modal operators, and whose semantic structures are over partial-orderings with linear interval property, i.e. every interval within partial ordering is linear.

The syntax of propositional interval temporal logics is constructed from the following: the set of *propositional variables*, the *truth values*, the *classical operators (boolean operators, negation, etc.)*, and a *set of temporal operators* defined for each logic.

6.1.1 Interval Structures

Assume T is a set of time points and $\mathbb{T} = \langle T, < \rangle$ is a strict partial ordering. An *interval* in $\mathbb{T}$ is defined as a pair $[t_1, t_2]$ with $t_1, t_2 \in T$. $[t_1, t_2]$ is called a *strict interval* if $t_1 < t_2$, and a *non-strict interval* if $t_1 \leq t_2$. Intervals of the form $[t_1, t_1]$ are called *point intervals*.

Let $\mathbb{I}(\mathbb{T})$ be a set of intervals on $\mathbb{T}$. We denote the *set of strict intervals* on $\mathbb{T}$ as $\mathbb{I}(\mathbb{T})^-$, and the *set of all (strict and point) intervals* on $\mathbb{T}$ as $\mathbb{I}(\mathbb{T})^+$. Also, we denote an *interval structure* as $\langle \mathbb{T}, \mathbb{I}(\mathbb{T}) \rangle$.

In this section we assume two different natural semantics for interval temporal logics: a *strict interval structure*, a tuple $\langle \mathbb{T}, \mathbb{I}(\mathbb{T})^- \rangle$, and a *non-strict interval structure*, a tuple $\langle \mathbb{T}, \mathbb{I}(\mathbb{T})^+ \rangle$. Both semantics have linear interval property (see Section 2).

6.1.2 The Logic HS

The logic HS (Halpern and Shoham) [17] is a relatively expressive logic, which is one of the mostly known propositional interval logics. Formulas are interpreted over intervals rather than time points. HS introduces the notion of *current interval*. All modal operators of HS are unary and qualitative, which can access the current and other intervals.

The formulas of HS are recursively defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle A \rangle \varphi \mid \langle B \rangle \varphi \mid \langle E \rangle \varphi \mid \langle \bar{A} \rangle \varphi \mid \langle \bar{B} \rangle \varphi \mid \langle \bar{E} \rangle \varphi$$

where $p \in AP$ (AP is a set of propositions). Informally speaking, if $\langle X \rangle \varphi$ holds at the current interval, where $X \in \{A, B, E, \bar{A}, \bar{B}, \bar{E}\}$, then for $X = A$, φ holds at some interval starting immediately after the current interval ends; for $X = B$, φ holds at some interval starting as the current interval starts and ending during the current interval; for $X = E$, φ holds at some interval starting during the current interval and ending as the current interval ends; for $X = \bar{A}$, φ holds at some interval ending just before the current interval starts; for $X = \bar{B}$, φ holds at some interval that the current interval starts with; and for $X = \bar{E}$, φ holds at some interval that the current interval ends with. We remark that Allen's all relations [18] between two distinct intervals can be expressed by means of these modal operators.

Given that $\langle \mathbb{T}, \mathbb{I}(\mathbb{T})^+ \rangle$ is a non-strict interval structure with linear interval property, a model $\mathcal{M}$ is a tuple $\mathcal{M} = \langle \mathbb{T}, \mathbb{I}(\mathbb{T})^+, L \rangle$, where $L : AP \mapsto 2^{\mathbb{I}(\mathbb{T})^+}$ maps each proposition into the set of intervals where it holds. The formal semantics of HS formulas is then defined as follows [17]:

$\mathcal{M}, [t_1, t_2] \models \langle A \rangle \varphi$ iff $(\exists t_3 : t_2 < t_3)$ s.t. $\mathcal{M}, [t_2, t_3] \models \varphi$
$\mathcal{M}, [t_1, t_2] \models \langle B \rangle \varphi$ iff $(\exists t_3 : t_1 \leq t_3 < t_2)$ s.t. $\mathcal{M}, [t_1, t_3] \models \varphi$
$\mathcal{M}, [t_1, t_2] \models \langle E \rangle \varphi$ iff $(\exists t_3 : t_1 < t_3 \leq t_2)$ s.t. $\mathcal{M}, [t_3, t_2] \models \varphi$
$\mathcal{M}, [t_1, t_2] \models \langle \bar{A} \rangle \varphi$ iff $(\exists t_3 : t_3 < t_1)$ s.t. $\mathcal{M}, [t_3, t_1] \models \varphi$
$\mathcal{M}, [t_1, t_2] \models \langle \bar{B} \rangle \varphi$ iff $(\exists t_3 : t_2 < t_3)$ $\mathcal{M}, [t_1, t_3] \models \varphi$
$\mathcal{M}, [t_1, t_2] \models \langle \bar{E} \rangle \varphi$ iff $(\exists t_3 : t_3 < t_1)$ $\mathcal{M}, [t_3, t_2] \models \varphi$

The logic HS has enough expressive power to distinguish different temporal structures, such as of *discrete*, *continuous*, *bound* or *linear* time structures. These are formally shown as follows [17]:

- $length0 \equiv [B]\bot$
- $length1 \equiv \langle B \rangle \top \wedge [B] length0$ ($length1$ holds at intervals with no proper subintervals.)
- $dense \equiv \neg length1$
- $discrete \equiv length0 \vee length1 \vee (\langle B \rangle length1 \wedge \langle E \rangle length1)$
- $unbound \equiv \langle A \rangle \top \wedge \langle \bar{A} \rangle \top$
- $linear \equiv (\langle A \rangle \varphi \rightarrow [A](\varphi \vee \langle B \rangle \varphi \vee \langle \bar{B} \rangle \varphi)) \wedge$
  $(\langle \bar{A} \rangle \varphi \rightarrow [\bar{A}](\varphi \vee \langle E \rangle \varphi \vee \langle \bar{E} \rangle \varphi))$

where $[X]\varphi$ is defined as $\neg\langle X \rangle \neg\varphi$ for $X \in \{A, \bar{A}, B, \bar{B}, E, \bar{E}\}$. Here, $length0$ informally says that the interval is actually a point; $length1$ asserts that there is no subinterval within the interval (i.e. there is no third point between the endpoints of the interval); *dense* asserts that subinterval exists (i.e. between any two points within the interval there is a third point); *unbound* asserts that for any time point with the interval we can find a predecessor and a successor; and *linear* asserts that any two points with the interval can be comparable.

HS is a suitable formalism to represent qualitative properties regarding continuous processes which assert continuous

change of time and make statements about events which require intervals rather than time points, e.g. 'a warning is received until the water level becomes normal'. The logic can also be used in reasoning about automatic planning; namely it can express properties regarding carrying out tasks, resource management, interacting with agents, etc. For example, the properties like "if a robot executes the charge-battery routine then at the beginning of the following execution of the navigate routine its batteries will be fully charged" [17] can be expressed in HS.

HS is a quite expressive logic due to its large modal operator set. However, it is not axiomatisable and is highly undecidable. The following theorems are taken from [17]:

- "The validity problem interpreted over any class of ordered structures with an infinitely ascending sequence is r.e.-hard (Thus, in particular, HS is undecidable for the class of all *(non-strict)* models, *linear* models, *discrete* linear models, *dense* linear models and *unbounded* linear models)."
- "The validity problem interpreted over any class of Dedekind complete ordered structures having an infinitely ascending sequence is $\prod_1^1$-hard (For instance, the validity in any of the orderings of the natural numbers, integers, or reals is not recursively axiomatisable. Undecidability even occurs in the classes of structures with no infinitely ascending sequences)."
- "The validity problem interpreted over any class of Dedekind complete ordered structures having unboundedly ascending sequences is co-r.e.-hard."

The undecidability results given above are based on the observation that HS formulas encode the computation of a Turing machine. In [141] undecidability was proved by means of a tiling problem.

In [142] some interesting results for the logic HS were presented. By using a geometrical representation for the modalities a sound and complete proof system for HS was introduced. [142] also proved that HS is a more expressive logic than any other temporal logic based on "linear orderings of time instants".

In [17] a translation machinery that converts an HS formula to its equivalent first-order formula on a corresponding first-order structure was provided. Such a translation is useful to reduce problems to well-known results in first-order logic.

In the literature some sublogics of HS are introduced. The logic BE is a fragment of HS containing the modal operators $\langle B\rangle$ and $\langle E\rangle$. BE can express the conditions on the underlying interval structure as with HS, except the formula *linear*. In [143] the satisfiability problem for BE formulas interpreted over all non-strict linear structures was found to be undecidable.

We cannot mention the other studied fragments of HS due to the space limitation. We refer the reader to the recently published survey [144] on decidable and undecidable fragments of the logic HS.

6.1.3 The Logic CDT

The logic CDT was introduced by Venema in [19]. It is one of the most expressive propositional interval logic over linear orderings [7]. CDT includes the binary modal operators C, D and T. The formal semantics over non-strict linear structures is defined as follows:

$\mathcal{M}, [t_1, t_2] \models \pi$ iff $t_1 = t_2$

$\mathcal{M}, [t_1, t_2] \models \varphi C \psi$ iff $(\exists t_3 : t_1 \leq t_3 \leq t_2)$ s.t. $\mathcal{M}, [t_1, t_3] \models \varphi$ and $\mathcal{M}, [t_3, t_2] \models \psi$

$\mathcal{M}, [t_1, t_2] \models \varphi D \psi$ iff $(\exists t_3 : t_3 \leq t_1)$ s.t. $\mathcal{M}, [t_3, t_1] \models \varphi$ and $\mathcal{M}, [t_3, t_2] \models \psi$

$\mathcal{M}, [t_1, t_2] \models \varphi T \psi$ iff $(\exists t_3 : t_2 \leq t_3)$ s.t. $\mathcal{M}, [t_2, t_3] \models \varphi$ and $\mathcal{M}, [t_1, t_3] \models \psi$

where $\mathcal{M}$ is a tuple $\mathcal{M} = \langle \mathbb{T}, \mathbb{I}(\mathbb{T})^+, L\rangle$. Informally speaking, $\varphi C \psi$ holds at the current interval, if it can be *chopped* into two intervals where φ holds at the first one and ψ holds at the second one; $\varphi D \psi$ holds at the current interval, if there exist two intervals I at which ψ holds and J at which φ holds such that the current interval starts during I and ends with I, and the current interval is met by J (i.e. the end of J is the beginning of the current interval); and $\varphi T \psi$ holds at the current interval, if there exist two intervals I at which ψ holds and J at which φ holds such that the current interval starts with I and ends during I, and the current interval meets J (i.e. the end of the current interval is the beginning of J).

These operators subsume all unary modalities of propositional interval logics of Allen's interval relations [145]:

$$\begin{array}{ll} \langle B\rangle \varphi \equiv \varphi\ C\ (\neg\pi) & \langle \overline{B}\rangle \equiv (\neg\pi)\ T\ \varphi \\ \langle E\rangle \varphi \equiv (\neg\pi)\ C\ \varphi & \langle \overline{E}\rangle \equiv \varphi\ D\ (\neg\pi) \\ \langle A\rangle \varphi \equiv (\neg\pi \wedge \varphi)\ T\ \top & \langle \overline{A}\rangle \equiv (\neg\pi \wedge \varphi)\ D\ \top \end{array}$$

CDT can distinguish different classes of temporal structures, such as discrete, continuous, bound, linear or complete time structures. For example, the discreteness of an interval can be specified in CDT as follows:

$$(\mathit{length}1\ C\ \top) \wedge (\top\ C\ \mathit{length}1).$$

Since CDT is more expressive than, all properties expressed in HS can also be expressed in CDT, e.g. qualitative

properties regarding continuous processes, statements about events which require intervals, automatic planning, etc.

[19] gives an axiomatic system which is sound and complete for the logic CDT which is interpreted over non-strict linear models. This axiomatic system can be extended for the classes of *discrete linear orderings, dense linear orderings*, etc. [7]. Since CDT subsumes HS, the satisfiability problem for "CDT is not decidable over almost all classes of linear orderings, including *discrete, dense, continuous*, etc." [7].

The partial-order semantics of CDT has been recently studied in [146], where the logic BCDT$^+$ is introduced. BCDT$^+$ uses the language of CDT with partial-order semantics of linear intervals. Finite axiomatizability of CDT and its single-modality fragment, and decidability of these possible fragments were open for a long time. Recently, [147] has shown that "almost all fragments of CDT, containing at least one binary operator, are neither finitely axiomatizable with standard rules nor decidable". This result fills an important gap in the spectrum.

6.1.4 The Logic PNL

Propositional Neighbourhood Logic (PNL) is the propositional fragment of First-Order Neighbourhood Logic introduced in [148]. It has been studied on both strict and non-strict linear structures in [149]. The language with non-strict semantics is called PNL$^{\pi+}$ including the modalities $\Diamond_r$ (*met by*, i.e. right neighbouring) and $\Diamond_l$ (*meets*, i.e. left neighbouring), and the model constant π (representing *point* intervals). The modal operators can have either strict or non-strict semantics.

Given that $\mathcal{M} = \langle \mathbb{T}, \mathbb{I}(\mathbb{T})^+, L \rangle$, the formal (non-strict) semantics of PNL$^{\pi+}$ formulas is then defined as follows:

$\mathcal{M}, [t_1, t_2] \models \pi$ iff $t_1 = t_2$
$\mathcal{M}, [t_1, t_2] \models \Diamond_r \varphi$ iff $(\exists t_3 \geq t_2)$ s.t. $\mathcal{M}, [t_2, t_3] \models \varphi$
$\mathcal{M}, [t_1, t_2] \models \Diamond_l \varphi$ iff $(\exists t_3 \leq t_1)$ s.t. $\mathcal{M}, [t_3, t_1] \models \varphi$

Assume PNL$^+$ denotes the non-strict PNL without the modal constant π, and PNL$^-$ denotes the strict PNL without the modal constant π . The logic PNL$^{\pi+}$ subsumes both PNL$^+$ and PNL$^-$ [145].

Given that formulas are interpreted over strict linear models, PNL$^-$ has enough expressive power to distinguish the different classes of linear structures, such as discreteness, continuity, boundness, or completeness. For example, unboundness and density can be specified in PNL$^-$ as follows [7]:

- *unbound* $\equiv \Box_r \varphi \rightarrow \Diamond_r \varphi$
- *dense* $\equiv (\Diamond_r \Diamond_r \varphi \rightarrow \Diamond_r \Diamond_r \Diamond_r \varphi) \wedge (\Diamond_r \Box_r \varphi \rightarrow \Diamond_r \Diamond_r \Box_r \varphi)$

PNL has a reach syntax, which is useful to specify qualitative properties regarding planning, natural language processing, digital systems etc. Here we give an example property from digital systems. The following PNL$^{\pi+}$ formula

$$(\neg\pi \wedge p) \rightarrow \Diamond_r(\neg\pi \wedge \Diamond_r(\neg\pi \wedge q))$$

asserts that "the output q of a device to strictly follow the input p" [149]. Here, p and q are constraint to the instantaneous states. For this reason, the intervals are not to be allowed be point intervals, which was done using the negation of the modal constant π.

In [149] several sound and complete axiomatic systems were provided for various classes of models. In addition to strict linear models [149] also provides sound and complete axiomatic systems for non-strict linear structures, complete unbounded linear structures, unbounded structures, dense structures, discrete structures, dense unbounded structures and discrete unbounded structures. As for decidability results, [150] shows that the satisfiability problem for PNL$^{\pi+}$, PNL$^+$ and PNL$^-$ over the integers is NEXPTIME-complete. [150] introduces a sound and complete tableau algorithm, and shows that it is optimal. In [151], the expressive power of PNL$^{\pi+}$, PNL$^+$ and PNL$^-$ is compared, and it is shown that PNL$^{\pi+}$ is strictly more expressive than PNL$^+$ and PNL$^-$. [151] proves that "the satisfiability problem for PNL$^{\pi+}$ over the class of all linear orders, as well as over some natural subclasses of it, such as the class of all well-orders and the class of all finite linear orders, can be decided in NEXPTIME by reducing it to the satisfiability problem for the two-variable fragment of first-order logic over the same classes of structures".

An important fragment of the PNL is the *Right Propositional Neighbourhood Logic* (RPNL) which is based on the right neighbourhood relation between intervals. The language with non-strict semantics is called RPNL$^{\pi+}$. The non-strict fragment without the modal constant π is denoted by RPNL$^+$, and the strict fragment without the modal constant π is denoted by RPNL$^-$. As for decidability results, in [152] an EXPSPACE tableau-based decision procedure is devised for RPNL$^-$ interpreted over natural numbers. In [153] another NEXPTIME decision procedure is developed. This method works for all classes of RPNL, which are RPNL$^{\pi+}$, RPNL$^+$, and RPNL$^-$, interpreted over natural numbers. [153] also proves the optimality of the decision procedure.

Recently, PNL has been extended with metric operators. [154] introduces the logic MPNL which extends PNL with a new operator $\texttt{len}_{\sim k}$ (where $\sim \in \{<, >, =, \leq, \geq\}$) which denotes a metric constraint for the current interval. For exam-

ple, $\texttt{len}_{=2}$ imposes the current interval to be have a length 2. Constraining the length of the intervals allows writing quantitive properties, such as the formula

$$\Diamond_r(\texttt{len}_{=k} \wedge \Diamond_r(\texttt{len}_{=0} \wedge p))$$

asserting that p holds at some future point within a distance k from the current interval. [154] show that MPNL is decidable in 2-EXPTIME and "expressively complete with respect to a well-defined sub-fragment of the two-variable fragment of first-order logic for linear orders with successor function, interpreted over natural numbers".

6.1.5 Subinterval Logics

For many years, the high computational complexity of interval logics (such as HS and CDT) restricted these logics in practical applications and semantic investigation. Recently, the trend has shifted to finding expressive decidable fragments. The most important decidable fragments are PNL and its fragments, logics of neighbourhood [155], *sub-interval* and *superinterval* structures. In particular, the logic of subintervals has received more attention. It was first studied in [20], where the subinterval relation $\subseteq$ is considered. The logic includes the modal operator $\langle D\rangle$, which allows looking inside the current interval [17]. The semantic definition of modal operator $\langle D\rangle$ is defined as follows:

$$\mathcal{M}, [t_1, t_2] \models \langle D\rangle \varphi \text{ iff } \exists [t_1', t_2'] \subseteq [t_1, t_2] \text{ s.t. } \mathcal{M}, [t_1', t_2'] \models \varphi.$$

$\langle D\rangle \varphi$ informally means that φ is true during the current interval. When the strict semantics is considered and formulas are interpreted over the rational numbers or the class of all linear orderings, the logic D becomes equivalent to the standard modal logic S4; when formulas are interpreted over integers, the logic D becomes equivalent to the modal logic S4 with the axiom $[D]([D](p \rightarrow [D]p) \rightarrow p) \rightarrow p$, expressing that $\subseteq$ is well-formed [145]. The satisfiability problem for both S4 and S4 with the above axiom is known to be PSPACE-complete ([156, 157]). In [158] the logics of subinterval structures over *dense* linear orders is shown to be decidable. [158] also provides a tableau-based decision procedure, which is shown to be PSPACE-complete. [159] shows that "the satisfiability problem for interval logics of the reflexive sub-interval and super-interval relations interpreted over *finite* linear orders is PSPACE-complete".

The decidability of subinterval logics depends not only on the choice of the domain, but also on the definition of the semantics of the operators. In this respect, some negative results have been published in [160], where the authors show that the logic D is undecidable when interpreted over the class of finite orderings and over the class of all discrete orderings. The undecidability results from [160] holds when the subinterval relation is unreflexive (i.e. $[t_1', t_2']$ is a strict subset of $[t_1, t_2]$) or when it corresponds to the Allen's relation "during" ($t_1 < t_1'$ and $t_2' < t_2$).

Subinterval logics have also been investigated in natural language discourse. In [161] a sub-interval logic, which is used in capturing temporal prepositions of a natural language, is introduced. In [162] a quantitative extension of this logic is represented. Both logics are decidable, and their satisfiability problems are in NEXPTIME.

6.2 First-Order Interval Temporal Logics

First-order interval temporal logics were originally defined to formally specify and verify hardware components of real-time systems. ITL is the most commonly known first-order interval temporal logic. Numerous extensions of ITL, such as Duration Calculus [163], Neighbourhood Logic [163] etc., have been introduced. Below we will review well-known first-order interval temporal logics.

6.2.1 The Logic ITL

ITL was first introduced in [13] (which was "interpreted over discrete linear orderings with finite time intervals" [7]). ITL syntax and semantics can be defined as follows [6, 13]:

The alphabet is constructed from the following: an infinite set of global variables $x, y, z, ...$, an infinite set of temporal variables $t, t', ...$, an infinite set of global function symbols $f^n, g^m, ...$, where f^n is a function of arity n and g^m is a function of arity m, an infinite set of predicate symbols $P^n, R^m, ...$, where P^n is a predicate of arity n and R^m is a predicate of arity m, and an infinite set of propositions $p, q, ...$. The set of terms θ is defined by:

$$\theta ::= x \mid t \mid f^n(\theta_1, ..., \theta_n)$$

The formulas of ITL can be recursively defined as follows:

$$\varphi ::= p \mid P^n(\theta_1, ..., \theta_n) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi^\frown\psi \mid (\exists x)\varphi$$

where $^\frown$ denotes the *chop* operator (similar to CDT's chop operator C). Let Δ be the set of temporal variables, AP be the set of temporal propositional letters and $\mathbb{I}(\mathbb{R})$ be the set of all bounded and closed intervals of real numbers $\{[t_1, t_2] : t_1 \leq t_2 \wedge t_1, t_2 \in \mathbb{R}\}$. The meanings of temporal variables and propositions, i.e. the interval-dependent symbols, are given by the interpretation:

$$\mathcal{J} \in (\Delta \rightarrow (\mathbb{I}(\mathbb{R}) \rightarrow \mathbb{R})) \cup (AP \rightarrow (\mathbb{I}(\mathbb{R}) \rightarrow \{\top, \bot\}))$$

where $\mathcal{J}(t)([t_1,t_2]) \in \mathbb{R}$ for all $t \in \Delta$, $\mathcal{J}(\ell)([t_1,t_2]) = t_2 - t_1$ (ℓ is a special temporal variable denoting the interval length), $\mathcal{J}(p)([t_1,t_2]) \in \{\top, \bot\}$ for all $p \in AP$.

A valuation is a mapping L which associates a real number with each global variable. Given a variable x, two valuations L and L' are said to be $x-$*equivalent* if $L(y) = L'(y)$ for every global variable y which is different from x.

Assume that a total function $\bar{f}^n \in \mathbb{R}^n \to \mathbb{R}$ is associated with each n-ary function symbol f^n. The semantics of a term θ at an interval $[t_1,t_2]$ under a valuation L is denoted by $\mathcal{J}^L_{[t_1,t_2]}(\theta)$. The function $\mathcal{J}^L_{[t_1,t_2]}$ is recursively defined as follows:

- for a global variable x, $\mathcal{J}^L_{[t_1,t_2]}(x) = L(x)$
- for a temporal variable t, $\mathcal{J}^L_{[t_1,t_2]}(t) = \mathcal{J}(t)([t_1,t_2])$
- for a term θ of the form $f^n(\theta_1,...,\theta_n)$, $\mathcal{J}^L_{[t_1,t_2]}(\theta) = \bar{f}^n(\alpha_1,...,\alpha_n)$

where $\alpha_i = \mathcal{J}^L_{[t_1,t_2]}(\theta_i)$ for $1 \le i \le n$.

Assume that a total function $\bar{G}^n \in \mathbb{R}^n \to \{\top, \bot\}$ is associated with each n-ary relation symbol G^n. Let $\mathcal{M} = \langle \mathcal{J}, L \rangle$ be a model for ITL. The formal semantics of ITL formulas is then defined as follows:

$\mathcal{M}, [t_1,t_2] \models G^n(\theta_1,...,\theta_n)$ iff $\bar{G}^n(\alpha_1,...,\alpha_n) = \top$ where $\alpha_i = \mathcal{J}^L_{[t_1,t_2]}(\theta_i)$ for $1 \le i \le n$

$\mathcal{M}, [t_1,t_2] \models \varphi C \psi$ iff $(\exists t_3 : t_1 \le t_3 \le t_2)\ \mathcal{M}, [t_1,t_3] \models \varphi$ and $\mathcal{M}, [t_3,t_2] \models \psi$

$\mathcal{M}, [t_1,t_2] \models (\exists x)\varphi$ iff $\mathcal{M}, [t_1,t_2] \models \varphi$ for some value assignment L' which is x-equivalent to L

To show ITL at work we formally specify the statement "the process p_2 finishes t seconds later the process p_1" in ITL as follows:

$$(\exists t_1, t_2)\ p_1(t_1) \wedge p_2(t_2) \wedge \ell = t_2 \wedge (\ell = t_1 \frown \ell = t).$$

This property informally says that t_1 and t_2 are two time points, where p_1 and p_2 holds, respectively, and the distance between t_2 and t_1 is t.

Not surprisingly ITL is highly undecidable. A sound and complete axiomatic system is represented in [164]. [164,165] consider some local variants of ITL, and provide sound and complete proof systems for ITL with the locality constraint. [166] provides a complete proof system for ITL extended with projection.

ITL's reach language provides a suitable notation for reasoning about time periods, which finds applications in hardware and software systems. Many temporal logics cannot deal with both sequential and parallel composition; but ITL offers a formal framework which allows dealing with both and provides "powerful and extensible specification and proof techniques for reasoning about properties involving safety, liveness and projected time" [167].

6.2.2 The Logic DC

Duration Calculus (DC) [163] is a first-order interval temporal logic with the additional notion of *state*, which is characterised by a *duration*[9].

DC is an extension of ITL in the sense that temporal variables other than ℓ have a structure $\int S$, where $\int S$ is called a *state duration* and S is called a *state expression*. The rest of the alphabet is same as ITL. The syntax and semantics of DC are defined as follows [6]:

The set of state expressions is constructed from a set of *state variables*:

$$S ::= 0 \mid 1 \mid P \mid \neg S_1 \mid S_1 \vee S_2$$

Let $\mathcal{S}$ be a set of state variables. The meanings of state variables, temporal variable ℓ, and propositional letters are given by the interpretation:

$$\mathcal{J} \in (\mathcal{S} \to (\mathbb{R} \to \{0,1\})) \cup (\{\ell\} \to (\mathbb{I}(\mathbb{R}) \to \mathbb{R})) \cup (AP \to (\mathbb{I}(\mathbb{R}) \to \{\top, \bot\}))$$

where $\mathcal{J}(S)(t) \in \{0,1\}$ for all state variables $S \in \mathcal{S}$ and $t \in \mathbb{R}$, $\mathcal{J}(\ell)([t_1,t_2]) = t_2 - t_1$, $\mathcal{J}(p)([t_1,t_2]) \in \{\top, \bot\}$ for all $p \in AP$.

Given the interpretation $\mathcal{J}$, the semantics of a state expression S is a total function $\mathfrak{I}[S] : \mathbb{R} \to \{0,1\}$ which has a finite number of discontinuity points only. For any time point t, the semantics can be defined inductively on the structure of state expressions as follows:

$$\mathfrak{I}[0](t) = 0$$
$$\mathfrak{I}[1](t) = 1$$
$$\mathfrak{I}[P](t) = \mathcal{J}(P)(t)$$
$$\mathfrak{I}[\neg S_1](t) = 1 - \mathfrak{I}[S_1](t)$$
$$\mathfrak{I}[S_1 \vee S_2](t) = \begin{cases} 0 \ if\ \mathfrak{I}[S_1](t) = 0 \ and\ \mathfrak{I}[S_2](t) = 0 \\ 1 \ otherwise \end{cases}$$

The semantics of a duration $\int S$ in a given model, with respect to an interval $[t_1,t_2]$, can be defined by

$$\mathfrak{I}\left[\int S\right]([t_1,t_2]) = \int_{t_1}^{t_2} \mathfrak{I}[S](t)\,dt.$$

We can define some useful abbreviations in DC:

$$\lceil S \rceil \equiv \ell = 0$$
$$\lceil S \rceil \equiv \int S = \ell \wedge \ell > 0$$

[9] "The duration of a state is the length of the time period during which the system remains in the state" [7].

All axioms and inference rules of ITL can be adopted for DC. However, additional axioms are needed for temporal variables.

DC has been used in the specification and verification of various complex systems. Due to its continuous and interval nature it provides more expressive power to specify the accumulated existence of a state over an interval. For example, the properties, such as within an interval of length ℓ the total presence of the state s is $1/3$ of length of the interval. This property cannot be expressed in any real-time logic we discussed above, such as MTL, RTL, etc. We show the formal specification of this property by considering a real-time requirement of a gas burner system. The requirement, "the proportion of leak time in any interval is not more than one-twentieth of the interval, if the interval is at least one minute long", is expressed in DC as follows [163]:

$$\Box(\ell \geq 60 \rightarrow 20 \int Leak \leq \ell).$$

All axioms and inference rules of ITL can be adopted in DC. However, additional axioms are needed for temporal variables. In [6] an axiomatic system for Duration Calculus is given. The satisfiability problem for both first-order and propositional DC is shown to be undecidable [168].

Several fragments of DC have been investigated so far. In [168] a fragment of propositional DC, called RDC, was introduced. It was shown that RDC has a decidable satisfiability problem when interpreted over $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$. In [169] the satisfiability problems of several extensions of RDC were studied. In [170] an extension of RDC was presented on continuous time "with a restriction on the finite variability such that the number of discontinuous points of any state in any unit interval has a fixed upper bound". In [171] a decidable variant of DC was presented, where negation is removed from the syntax; but an iteration operator is introduced together with some form of inequalities. In [172] another fragment of propositional DC, which can capture Allen's relations [18], was introduced by imposing some syntactic restrictions. By proposing a sound, complete and terminating decision algorithm, it was shown that the satisfiability problem is decidable. In [173] a logic with quantification over states was introduced. It was shown that the satisfiability of formulas is decidable. In [148] Duration Calculus and first-order neighbourhood logic were combined, and a axiomatic systems for DC and NL were merged. It was proved that "the fragment of DC/NL obtained by restricting the formulas" of state expressions is decidable [7]. An extension with formulas with equality becomes undecidable.

Model checking problem for DC is a challenging task. In general, there has not been a general model checking technique for this logic. To have efficient model checking techniques, it is necessary to consider a fragment of the logic. In [170] some model checking tools were developed for a class of models which are restricted to some possible behaviours of real-time systems. In [174–177] some techniques were developed to check if a timed automaton satisfies a formula of the type "linear duration invariants". In [178] some algorithms were developed to check the satisfiability over integer models. In [173] a DC validity checker, called DC-VALID, to check the satisfiability of formulas which are interpreted over discrete-time. [179] suggested *bounded validity checking* [180] of "a discrete-time DC without timing constraints by polynomial-sized reduction to propositional SAT solving". In [181] a decidability result and a model-checking algorithm are presented "for a rich subset of DC through reductions to first-order logic over the real-closed field and to multi-priced timed automata (MPTA)".

6.2.3 The Logic NL

Although DC is a very expressive logic, it has a limitation that it does not allow to reason about outside of the current interval. The logic NL, proposed in [163], solves this problem. It replaces DC's C operator with the *left neighbourhood* modality $\Diamond_l$ and *right neighbourhood* modality $\Diamond_r$, which allow us to look outside of the interval.

The semantics of the modal operators $\Diamond_l$ and $\Diamond_r$ is defined as in PNL, and the rest of the semantics is defined as in DC. NL can express any of the Allen's interval relations; thus, it can represent important properties, such as discreteness, density, boundedness, etc; for example, the chop operator $\frown$ can be expressed in terms of the modalities $\Diamond_l$ and $\Diamond_r$ as follows [7]:

$$\varphi \frown \psi = \exists x, y\,(\ell = x + y) \wedge \Diamond_l \Diamond_r\,((\ell = x) \wedge \varphi \wedge \Diamond_r\,((\ell = y) \wedge \psi)).$$

NL is a highly expressive logic, which can specify many interesting real-time aspects such as concurrency, scheduling, shared resources, periodic behaviour, which many logics cannot express. Here, we give an example (from [182]) to show how NL expresses complex scheduling properties, such as

$$\Box(\ell \geq T \rightarrow (\int p \leq (1 - x).\ell))$$

which asserts that the server provides a service to the process p for at least a fraction x of the time, if the server runs for at least T time units.

NL is an undecidable logic like DC. In [183] a sound and complete axiomatic system is given for the logic NL. In [184] the *up* and *down* modalities, represented by $\Diamond_u$, $\Diamond_d$ respectively, were introduced, and a two dimensional version of NL was proposed.

6.2.4 The Logic IDL

Duration Calculus is a very expressive logic for specifying real-time requirements; but the automata theory for DC models is rather primitive and there are no available tools. By contrast, the state sequences with time has been widely used in real-time system behaviour [185]. The automata theory of timed state sequences have been applied to tools such as Hytec [186], Uppaal [187], Kronos [188] etc.

[189] introduced *Interval Duration Logic (IDL)*, which is defined on *timed state sequence* models and incorporates formulas with *cumulative* amount of time. Due to its expressive syntax it can express complex real-time properties, e.g. scheduling and planning constraints. As an example, we give a specification example from a gas burner system. The property 'between two instances of Leak there is at least k seconds' is specified in IDL as follows:

$$\Box((\lceil Leak \rceil^\frown \lceil \neg Leak \rceil^\frown \lceil Leak \rceil) \Rightarrow \ell \geq k)$$

IDL is a very expressive logic, but it is undecidable. However, there are some methods which have been proposed for the satisfiability and model checking problems of IDL. [190] applies *bounded validity checking* techniques [180] to IDL "by polynomially reducing this to checking unsatisfiability of *lin-sat* formulae". [190] also compares various methods for the satisfiability problem "including digitization technique [191], combined with an automata-theoretic analysis [173]; digitization technique [191] followed by pure propositional SAT solving [179]; and (c) lin-sat solving [192]".

[189] presents a decidable subset of IDL, which has a restriction that only *located time constraints* are allowed. The paper shows that the models of this subset can be considered as "timed words accepted by a finite state event-recording integrator automaton", which implies the satisfiability of the subset. It is also shown that the defined subset and event-recording automata have the same expressive power, which makes this logic an important decidable subset in the domain of DC.

7 Probabilistic Logics

Probabilistic reasoning has been the subject of computer science for a long time. There is an extensive study about formal systems with uncertainty. Probabilistic logics are used to formally specify and verify *dependability requirements*, expressing that the probability of system behaviour should be below a certain threshold [193], as well as functional requirements (depending on the syntax of the logic).

There are two main approaches: extending classical logic with probabilistic operators (such as modal logic of knowledge in [194]); combining a probabilistic approach with non-classical logics (such as the probabilistic extension of intuitionistic logic [195]). Below we review well-known probabilistic temporal logics.

7.1 Probabilistic Temporal Logics

7.1.1 The Logics PCTL and PCTL*

Probabilistic Computation Tree Logic, *PCTL* [196, 197], is a probabilistic extension of the branching time temporal logic CTL. PCTL replaces the CTL path quantifiers with probabilistic operators. Namely, it replaces the path formulas $\exists\varphi$ and $\forall\varphi$ with the probabilistic formula $\mathrm{P}_{\sim r}[\varphi]$, where $0 \leq r \leq 1$ and $\sim \in \{<, >, \leq, \geq, =\}$. The semantics of the probabilistic operator is defined as follows:

$$\mathcal{M}, s \models \mathrm{P}_{\sim r}[\varphi] \text{ iff } \pi_m(\sigma \in Paths(s) \; s.t. \; \mathcal{M}, \sigma \models \varphi) \sim r.$$

PCTL is interpreted over discrete-time Markov chains. The execution of a Markov chain constructs a set of *paths*, which are infinite sequences of states. The semantics of the probability operator P refers to the probability for the sets of paths for which a path formula holds. Namely, a probability measure π_m for the set of paths σ (starting from s) with a common prefix of length n, $s \rightarrow s_1 \rightarrow ... \rightarrow s_n$, is defined to be the product of transition probabilities along the prefix, i.e. $\mu((s_0, s_1)) \times ... \times \mu((s_{n-1}, s_n))$ [197]. $\mathrm{P}_{\sim r}[\varphi]$ informally means that φ holds for a set of paths σ from s with a probability $\sim r$.

Universal and existential quantification over paths is a subset of probabilistic quantification. For example, the existential path quantification $\exists$ can be represented by the probabilistic operator $\mathrm{P}_{>0}$. Therefore, PCTL's probabilistic operator provides a more general quantification, because as well as expressing a property is true at all/some paths, we can also express a property is true at more than 50% of the paths.

PCTL is very convenient to specify so-called *soft deadline properties*, e.g. "after a request for a service, there is at least a 98% probability that the service will be carried out within 2 seconds" [197]. Soft deadline properties are important in real-time system specification. As an example, the property that "the probability of φ eventually occurring is greater than or equal to r" can expressed in PCTL as follows:

$$\mathrm{P}_{\geq r}[true\ \mathcal{U}\ \varphi]\,.$$

[197] presents a model checking algorithm for PCTL, which is polynomially bounded by the size of the formula and the *Markov chain*[10] model. [198] analyses the decidability problem of PCTL. The authors show that if the probability constraint r is restricted to only 0 and 1, then the decidability problem becomes EXPTIME-complete; on the other hand, the decidability problem for arbitrary r values between 0 and 1 is still open.

[199] defines another probabilistic variant of CTL, namely a probabilistic extension of CTL* [69]. This new logic is called *PCTL**, which can specify quantitative probabilistic properties of systems, e.g. quantitative bounds on the probability of system evolutions, modelled as discrete *Markov processes*[11]. [69] also extends discrete Markov processes to *generalized Markov processes*[12], where transition probability function is not total. Generalized Markov processes are convenient to model ''abstraction" and "refinement". [199] also presents an elementary model checking algorithm for PCTL* over discrete Markov processes, which is then extended for generalized discrete Markov processes. This algorithm can also be used to determine the satisfiability of PCTL* formulas. In fact, [199] shows that the decision problem for PCTL* formulas on generalized Markov processes is decidable. However, no efficient computational method is given for this problem. In addition, no sound and complete axiomatisation of the logic is given.

PCTL* provides a richer syntax than PCTL, because it also employs LTL semantics. For example, PCTL* formula

$$\mathrm{P}_{\geq r}[\Box(true\ \mathcal{U}\ \varphi)]\,.$$

asserts that the probability of φ occurring infinitely often is greater than or equal to r" cannot be expressed in PCTL.

[200] shows that model-checking algorithms for extensions of PCTL and PCTL* to probabilistic-nondeterministic models have a polynomial-time complexity in the size of the model, which is same as the model checking complexity on Markov chains [196, 197, 199]. This result shows that adding nondeterminism does not increase model checking complexity in the size of the model. When we consider time bounds expressed in terms of the size of the formula, the situation is different. The model checking complexity of PCTL is linearly bounded in the size of the formula for both Markov chains and probabilistic-nondeterministic systems. However, while model checking complexity of PCTL* on Markov chain is exponentially bounded in the size of formula, it is in double exponential time on probabilistic-nondeterministic systems.

7.1.2 The Logic PTCTL

A probabilistic extension of the real-time branching logic TCTL is defined in [201]. The logic is called *PTCTL*, which combines both the logics TCTL and PCTL. PTCTL syntax is similar to PCTL; but the semantics is entirely different. Since it employs real-time, the semantics of the path operators $\mathcal{U}^{\sim k}$ is defined as in TCTL (see Section 3). On the other hand, the path operators of PCTL is interpreted over discrete time.

PTCTL can formalise properties such as 'with a probability higher than 0.9 the message is delivered within 5 seconds'. This can be expressed in PTCTL as follows:

$$\mathrm{P}_{>0.9}[true\ \mathcal{U}^{\leq 5}\ rcv].$$

PTCTL's rich language allows us to express very important real-time properties which cannot be expressed by real-time and interval logics, such as uncertainty, unpredicted behaviour e.g. system failure, communication and synchronization protocols (with random, continuously distributed delay), etc. [202].

Since PTCTL is a probabilistic extension of TCTL, PTCTL is also an undecidable logic. [201] shows that the model checking problem is "polynomial in the size of region graph and linear in the size of formula". It follows that the model checking problem is EXPTIME due to the size of region graph, which is exponential in the size of the model. [203] shows that the model checking problem is EXPTIME-complete. [203] also shows that the model checking problems of the subclass of PTCTL without punctual timing and the subclass which restricts the probability constraint to only 0 and 1 are PTIME-complete.

---

[10] A *Markov chain* is a tuple (S, P) where S is a set of *states* and $P : S \times S \rightarrow [0,1]$ is the *transition probability matrix* such that $(\forall s \in S) \sum_{s' \in S} P(s, s') = 1$.

[11] A (finite) *Markov process* is a 4-tuple $(AP, S, P, \mathcal{L})$, where AP is a finite set of *atomic propositions*, S is a countable nonempty set of *states*, $P : S \times S \rightarrow [0,1]$ is the *transition probability function* such that $(\forall s \in S) \sum_{s' \in S} P(s, s') = 1$ and $\mathcal{L} : S \rightarrow 2^{AP}$ is a *labeling function* [199].

[12] A *generalized Markov process* is a 3-tuple $(AP, S, \mathcal{L})$ (where AP, S and $\mathcal{L}$ are defined as in Markov processes) and a finite set of constraints on the transition probabilities [199].

7.1.3 The Logic PLTL

A propositional probabilistic discrete-linear temporal logic, called Probabilistic Propositional Temporal Logic (*PLTL*), is introduced in [204]. PLTL allows probabilistic reasoning, which is extended with temporal aspects. The logic is interpreted over linear time points, and includes a probabilistic operator as well as standard temporal operators, such as 'next', 'until', 'sometime' and 'always'. The probabilistic operator quantifies events along a single time line, therefore it is possible to express sentences such as "(according to the current set of information) the probability that sometime in the future α is true is at least n" [204].

Given that $\bigcirc$, $\Diamond$ and $\Box$ are the 'next', 'sometime' and 'always' operators, respectively, and $\mathrm{P}_{\sim r}$ ($\sim \in \{<, \leq, =, \geq, >\}$) is a probabilistic operator, an example of a PLTL formula is [204]

$$\bigcirc \mathrm{P}_{\geq r}[p] \wedge \Diamond \mathrm{P}_{<s}[p \rightarrow q] \rightarrow \Box\,(\mathrm{P}_{=t}[q])$$

which asserts "if the probability of p in the next moment is at least r and sometime in the future q follows from p with the probability less than s, then the probability of q will always be equal to t" [204].

[204] analyses completeness, decidability and complexity of the logic PLTL. It describes a class of so-called 'measurable models'. It is proved that "PLTL restricted to the class of all measurable models ($PLTL_{Meas}$)" has a sound and complete (infinitary) axiomatisation. The term infinitary means that the language and formulas are finite, but proofs can be infinite (Completeness cannot be proved with finitary axiomatisation). [204] shows that "a PLTL_{Meas}-satisfiable formula is satisfiable in an ultimately periodic model in which various parameters are bounded by functions depending on the size of the formula". [204] also shows that "the satisfiability problem for PLTL_{Meas} is PSPACE-hard, and that it belongs to NEXPTIME".

[204] also introduces First-order Probabilistic Temporal Logic (*FOPLTL*), which is the first-order version of PLTL. The complete infinitary axiomatisation is extended for the logic FOPLTL (No complete finitary axiomatisation is possible). The set of all FOPLTL-valid sentences is not recursively enumerable [42].

7.1.4 The Logic pDC

The Probabilistic Duration Calculus (*pDC*) [205] is an extension of Duration Calculus [163] with probabilities. pDC allows us to reason about probabilistic systems, and enables to express requirements such as a property holds with a certain probability. In pDC the system model is described as a finite automaton with fixed transition probabilities, which actually defines a discrete Markov process. The main idea as described in [205] is to express properties in DC, define satisfaction probabilities for formulas, and define a calculus to compute the probability of a formula from its subformulas' probabilities.

pDC satisfiability is described in [205] as follows: "Consider some finite probabilistic timed automata $\mathcal{A}$. The behaviours of $\mathcal{A}$ can be represented as a set of $\mathcal{M}$ of DC models. The probabilistic principles that manage the working of $\mathcal{A}$ used to introduce probability on the subsets of $\mathcal{M}$. Given a DC formula φ, the term

$$\pi(\varphi)(t)$$

denotes the probability of those models from $\mathcal{M}$ that satisfy φ at the interval $[0, t]$. A term of this sort is the component of pDC language" [205]. The term $\pi(\varphi)(t)$ clearly combines the probability and real-time aspects and can express the satisfaction probability of a requirement with real-time constraints, which cannot be expressed in pure DC. An example pDC formula is given below:

$$\pi_{s_0}((true; \lceil s \rceil); (\lceil s' \rceil; true))(t) = 0$$

which asserts that s' cannot immediately follows s. In [205] pDC is interpreted over discrete time; i.e. discrete transitions are assumed in models, defined as probabilistic time automata. In a later work, [193], pDC was defined for the case of continuous time, in which transitions in probabilistic automata take place in continuous time. In this logic, properties are written in terms of DC formulas. "Implementations of given requirements are modelled by continuous semi-Markov processes with finite space, which are expressed as finite automata with stochastic delays of state transitions (such an automaton is called continuous time probabilistic automaton)" [193]. [193] also defines a probabilistic model for DC formulas and a set of axioms/rules to calculate the satisfaction probabilities of DC formulas with respect to probabilistic automata. To our best knowledge, there is no complete proof system for this logic. pDC is, not surprisingly, an undecidable logic.

[206] defines the logic *Probabilistic Duration Calculus (PDC)*, which is another probabilistic extension of Duration Calculus. PDC extends DC syntax with formulas of type

$$\mathrm{P}_{\sim r}[\varphi]$$

where φ is a DC formula and $\sim \in \{>, \geq\}$. To show a formal specification example in PDC we extend the formal specification of the real-time requirement of a gas burner system as

follows: the probability that "the proportion of leak time in any interval is not more than one-twentieth of the interval if the interval is at least one minute long" is greater than and equal to 0.95. This requirement is expressed in PDC as follows:

$$\mathrm{P}_{\geq 0.95}[\Box(\ell \geq 60 \rightarrow 20 \int Leak \leq \ell)].$$

The syntax of PDC allows us "to reason about the probability of the satisfaction of a duration formula by a probabilistic timed automaton as well as to specify real-time properties of the system itself". PDC is interpreted over *behavioural models*[13], proposed in [208], which are a variant of probabilistic timed automata. [206] proposes a model checking technique which is an extension of the technique introduced in [177] "to check if a timed automaton satisfies a DC formula in the form of *linear duration invariants* or discretisable DC formulas based on searching the integral reachability graph of the timed automaton" [206] . The model checking problem is decidable "for a class of PDC formulas of the form *linear duration invariants*, or a formula for bounded liveness" [206].

7.1.5 The Logic PNL

[209] introduces the Probabilistic Neighbourhood Logic (*PNL*), which extends Neighbourhood Logic. [209] provides a complete proof system by extending the proof system of NL. In PNL, a more generalised version of probabilistic timed automata (defined in [193]) is assumed.

PNL has a similar grammar to the logic NL. It contains duration operators and probabilistic operators. The function symbols take a duration as argument and return a term of the probability. We now consider an example. Let φ denote a formula which is true at any interval between two consecutive processes. The following formula expresses "the assumption that the probability for the duration of such a period to be no bigger than x is a function of x which is the interpretation of the function symbol f in the model" [210]:

$$\mathrm{P}[\varphi \wedge \ell \leq x, x] = f(x).$$

PNL has the same expressive power as PDC, except for state expressions and their durations. Therefore, PNL can express dependability requirements and functional requirements expressible in NL. Since PNL is an extension of NL, it is an undecidable logic.

There are other probabilistic extensions of first-order interval logics, such as the probabilistic extension of ITL [211]; but they did not attract much attention, we therefore do not cover them in this paper.

7.2 Probabilistic Dynamic Logics

Another class of logics reasoning about *actions*, which are events causing the changes in the state of the world, and computer programs is *dynamic logics*. The early studies of these logics go back to the time when temporal logics started being studied in computer science. Temporal logics can reason about temporal evolution of the system state; but they cannot express changes as a result of applying actions. Dynamic logics allow referring to actions explicitly and describing changes caused by actions.

One of the most widely known dynamic logics is propositional dynamic logic (PDL), which includes the modalities 'sequence', 'choice', 'iteration' and 'test' to model program constructs. [212] gives a complete axiomatisation of PDL. [213] shows that the decidability problem is EXPTIME-complete. Other variants of PDL have also been introduced, such as Deterministic PDL, where PDL formulas are interpreted over deterministic structures, Strict PDL, where only deterministic *while* programs allowed, and strict deterministic PDL, where both deterministic and strict restrictions are applied. For a detailed and extensive discussion of dynamic logics and its variants, we refer the reader to [214].

Actions have also been considered within temporal logics. Here we give a few examples: [35] introduces an interval temporal logic to reason about actions and events, [215] proposes a temporal logic of actions to specify concurrent programs, and [216] extends LTL to reason about actions and programs. There are also numerous studies on action and change: [217–220] are only a few to name. We refer the reader to [221] for a detailed discussion on actions and agents.

Since Kozen's definition of formal semantics of probabilistic programs [222], some work has been done in this direction. Several systems have been introduced to formally study probabilistic programs. In particular, *probabilistic dynamic logics*, used to reason about properties and principle of probabilistic programs, received considerable attention. A brief historical development in this area is given below:

In [223], Feldman and Harel introduced a first-order prob-

[13] A *behavioural model* is a variant of probabilistic timed automata, where probabilistic transitions are discrete. "To resolve the nondeterminism between the passage of time and discrete transitions they use the concept of *adversary* which is essentially a deterministic schedule policy. Then, the set of executions of a probabilistic time automaton according to an adversary forms a Markov chain, and hence the satisfaction of a probabilistic CTL formula by this set can be defined, and then based on the region graph of the timed automaton the satisfaction of a probabilistic CTL formula by the timed automaton can be also verified" [207].

abilistic dynamic logic, called *Pr(DL)*, which can express properties of probabilistic programs. The syntax of this logic is similar to that of Pratt's first-order dynamic logic [224]. We can denote how probabilities occur in dynamic logics with an example. The following formula:

$$\mathrm{P}(true) = 1 \Rightarrow \{\alpha\}\mathrm{P}(true) \geq r$$

asserts that a program α ends with probability at least r. We refer the reader to [223] for the details. The semantics of Pr(DL) is based on extension of Kozen's formal semantics of probabilistic programs [222]. [223] provides a complete proof system for Pr(DL) relative to first-order analysis. [223] shows that for discrete probabilities the logic reduces to first-order analysis with integer variables. Since the underlying theory is highly undecidable, the logic Pr(DL) is also undecidable.

[225] defines the logic P-Pr(DL), which is a propositional fragment of the first-order dynamic logic Pr(DL). P-Pr(DL) has many important characteristics of Pr(DL), such as "the ability to use full first-order real-number theory for dealing with probabilities, and deterministic regular programs, while still being decidable" [225]. Neither the complexity of the decision procedure, nor a proof system is provided.

In [226] a probabilistic analog *PPDL* of Propositional Dynamic Logic is introduced. [226] proves the finite model property by showing that models can be reduced to an equivalent finite model with a bound on the number of states. A polynomial-space decision procedure is given to decide the validity of programs. [226] also provides "a deductive calculus" and shows its usefulness on an example program.

In [227] a Propositional Dynamic Logic with *explicit* probabilities is introduced. The language allows formulas of propositional probabilistic programs, where probabilistic operators are applied in a limited form. [227] provides a 2-EXPSPACE decision procedure for the logic by reducing it to "the decision problem of the theory of real closed fields".

[228] introduces a family of propositional calculi of qualitative probabilities (*QP*) with one binary operator $\leq$, which intuitively means "at least as probable as". Given that φ and ψ are two arbitrary QP formulas, $\varphi \leq \psi$ means that "the probability of φ is not greater than the probability of ψ" [229]. [228] presents a complete deductive system for QP, and shows that QP is decidable.

[229] extends QP with "many $\leq$-operators and operations among them that are analogous to the operations of composition, union, and iteration on modal operators known in propositional dynamic logic". The resulting logic (*DQP*) allows us to reason about probabilistic processes. The formula $w \models \varphi \leq_t \psi$ intuitively means that "the probability for a transition (experiment) t to transform w into a possible world that satisfies φ is smaller or equal to the probability for t to transform w into a possible world that satisfies ψ" [229]. An ω-complete proof system is presented for DQP in [229], which requires the building an infinite canonical model. This implies that DQP is undecidable.

7.3 Probabilistic Mu-Calculus

[230] presents the logic *Generalised Probabilistic Logic (GPL)*, which is a Mu-Calculus-based modal logic, in order to reason about "reactive probabilistic labelled transition systems (RPLTSs)". An RPLTS structure includes (probabilistic) transitions and (nonprobabilistic) actions, where nonprobabilistic actions are chosen *externally*, in contrary to Markov decision processes where nonprobabilistic choices are done *internally*.

GPL can be considered as a framework to define temporal logics on reactive models. GPL is an expressive logic. It provides a formal framework for reactive systems which includes probabilistic element regarding action and execution choices. Such systems include distributed systems, communications systems, etc. Some standard probabilistic (modal/temporal) logics, such as PLTL, PCTL, PCTL* etc., are subsumed by the logic GPL. For example, the PCTL formula

$$\mathrm{P}_{\geq 0.5}[true\ \mathcal{U}\ \varphi]$$

can be encoded in GPL as follows:

$$\mathrm{P}_{\geq 0.5}[\mu X.(\varphi \vee \langle a \rangle X)]$$

where μ denotes the *least fixed points* expressing for some states of the execution path and $\langle a \rangle \psi$ holds of an observation if there is an $a-$transition leading to the satisfaction of ψ (here a is an action).

[230] presents a model-checking algorithm which employs techniques to solve non-linear equations.

8 Conclusion

In this paper we have analysed various temporal formalisms, including propositional/first-order linear temporal logics, branching temporal logics, interval temporal logics, real-time temporal logics and probabilistic temporal logics. We have shown which aspects of real-time systems that these logics can express by providing various specification examples. In

Table 1: Specification of the gas burner requirement *any leak should not last more than 4 seconds* in different logics. L represents *Leak*.

Logic	Formal Specification
TPTL	$\Box x_1.\,(L \rightarrow L\,\mathcal{U}\; x_2.(\neg L \wedge x_2 \leq x_1 + 4))$
RTPLTL	$\Box\left(\left(\overline{L}^{*}\,L\right)true \rightarrow \left(\overline{L}^{*}\,L\right)^{\leq 4}(\neg L)true\right)$
CLTL	$\Box\left(L \rightarrow L\,\mathcal{U}_{[\sharp L \leq 4]}\,(\neg L)\right)$
MTL	$\Box\,(L \rightarrow L\,\mathcal{U}_{[0,4]}\,(\neg L))$
RTCTL	$\forall\Box\left(L \rightarrow \forall\left(L\,\mathcal{U}^{\leq 4}\,(\neg L)\right)\right)$
TCTL	$\forall\Box\left(L \rightarrow \forall\left(L\,\mathcal{U}^{\leq 4}\,(\neg L)\right)\right)$
TPCTL	$\forall\Box\left(L \rightarrow \forall\left(L\,\mathcal{U}_{\geq 1}^{\leq 4}\,(\neg L)\right)\right)$
CCTL	$\forall\Box\left(L \rightarrow \forall\left(L\,\mathcal{U}_{[\sharp L \leq 4]}\,(\neg L)\right)\right)$
RTL	$\forall i\,(@\,(\uparrow L, i) \leq @\,(\downarrow L, i) + 4)$
RTTL	$\Box T\,(L \wedge t = T \rightarrow L\,\mathcal{U}\,(\neg L) \wedge t \leq T + 4)$
RTIL	$\Box\,(\odot L^{+} \rightarrow L)\,(\mathtt{duration} \geq 4)$
TRIO	$\forall(t > 0 \wedge \mathrm{Futr}(L, t) \rightarrow \exists t'(t < t' \leq t + 4 \wedge \mathrm{Futr}(\neg L, t') \wedge \forall t''(t < t'' < t' \rightarrow \mathrm{Futr}(L, t''))))$
TILCO	$(L \rightarrow \neg L\;?\;[0, 4])\;\;@\;\;[0, +\infty)$
MPNL	$[G]\,(\neg\,(\mathtt{len}_{>4} \wedge L))$
ITL	$\forall t_1 \exists t_2\; L(t_1) \wedge \neg L_{(}t_2) \wedge \ell = t_2 \wedge ((\ell = t_1)\;\widehat{}\;(\ell \leq 4))$
DC, NL, IDL	$\Box\,(\lceil L \rceil \rightarrow \ell \leq 4)$
PTCTL	$\mathrm{P}_{\geq 1}\Box\left(L \rightarrow \mathrm{P}_{\geq 1}\left(L\,\mathcal{U}^{\leq 4}\,(\neg L)\right)\right)$
PDC	$\mathrm{P}_{\geq 1}\Box\,(\lceil L \rceil \rightarrow \ell \leq 4)$

order to show how a real-time system property can be expressed in different logics, we now use the gas burner design requirement [6] as a running example. Namely, we formally specify the property *any leak should not last more than 4 seconds*. Table 1 shows different formalisms of this real-time requirement using different logics. Note that since this property is a quantitative functional property, we only show the logics which can express this type of properties, i.e. the logics which employ metric operators.

Remark 1: The formal specifications of some logics can be syntactically very similar; but their semantics might be actually very different. For example, RTCTL and TCTL formulas of the property analysed are same, but these two formulas are represented by very different models. The former is interpreted over discrete structures, but the latter is interpreted over dense structures, which results in significantly different theoretical implications.

Remark 2: In Table 1, we have not included the logics which do not employ metric operators. In some cases, metric operators can be simulated by modal operators. For example, this property can be expressed in LTL as follows:

$$\Box(L \rightarrow L\mathcal{U}(\neg L \wedge \bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\neg L))$$

Such translations in general cause exponential blow-up in the size of formula, and therefore are not practical. In many cases, the translation is even not possible at all.

Remark 3: The bound constraints in some of the above formulas actually restrict the length of the duration that *Leak* occurs to less than 4. For example, in discrete-time logics the constraint '≤ 4' forces *Leak* to hold maximum 3 seconds. In order not to disturb the unity of the formulations, we ignore this type of minor oversights.

In the paper, we have also summarised important results on decidability, axiomatizability, expressiveness, model checking, etc. for each logic analysed, whenever possible. For a comparison of features of the temporal logics we discussed see Table 2, which is an extension of the one presented in [1]. Note that we use the following abbreviations: *No**: Undecidable in general, but decidable for some fragments or specific cases; *No***: No deduction system in general, but available for some fragments or specific cases; *No****: No model checking algorithm in general, but available for some fragments or specific cases; *Yes**: Decidable for some time domains; *Yes***: Available for some time domains; *Yes****: Available for some time domains.

Table 2

Logic	Logic Order	Fund. Entity	Temp. Struc.	Metric for Time	Decidability	Deductive Sys.	Model Checking
LTL	Propositional	Point	Linear	No	Yes	Yes	Yes
FOTL	First-order	Point	Linear	No	No*	No**	?
CTL	Propositional	Point	Branching	No	Yes	Yes	Yes
CTL*	Propositional	Point	Branching	No	Yes	Yes	Yes
CTL*[P]	Propositional	Point	Branching	No	Yes	Yes	Yes
TCTL	Propositional	Point	Branching	Yes	No	?	Yes
RTCTL	Propositional	Point	Branching	Yes	Yes	?	Yes
TPCTL	Propositional	Point	Branching	Yes	Yes	?	Yes
HS	Propositional	Interval	Linear	No	No	No	No
CDT	Propositional	Interval	Linear	No	No	Yes	No
PNL	Propositional	Interval	Linear	No	Yes	Yes	No
ITL	First-order	Interval	Linear	No	No	Yes	No
NL	First-order	Interval	Linear	Yes	No*	Yes	No
DC	First-order	Interval	Linear	Yes	No*	Yes	No***
IDL	First-order	Interval	Linear	Yes	No*	No	No***
RTL	First-order	Interval	Linear	Yes	No*	No	No***
RTIL	Propositional	Interval	Linear	Yes	Yes	No	?
RTTL	First-order	Point	Linear	Yes	No	Yes	No
TPTL	Propositional	Point	Linear	Yes	Yes*	Yes**	Yes***
MTL	Propositional	Point	Linear	Yes	Yes*	Yes	Yes***
MTIL	Propositional	Interval	Linear	Yes	Yes	?	Yes
XCTL	Propositional	Point	?	Yes	Yes*	?	Yes***
TRIO	First-order	Point	Linear	Yes	No	Yes	No***
TILCO	First-order	Interval	Linear	Yes	No*	Yes	No***
PCTL	Propositional	Point	Branching	No	Yes*	?	Yes
PCTL*	Propositional	Point	Branching	No	Yes*	?	Yes
PLTL	Propositional	Point	Linear	No	No*	No**	No
PDC	First-order	Interval	Linear	Yes	No	?	?
PNL	First-order	Interval	Linear	Yes	No	Yes	?

Acknowledgements.

This work was partially supported by EPSRC research project EP/F033567. Some parts of this paper exists in [231].

References

1. P. Bellini, R. Mattolini, and P. Nesi. Temporal logics for real-time system specification. *ACM Computing Surveys*, 32(1), 2000.
2. R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.
3. J. S. Ostroff. Formal methods for the specification and design of real-time safety critical systems. *Journal of Systems and Software*, 18(1):33–60, 1992.
4. P. Øhrstrøm and P. F. Hasle. *Temporal Logic: From Ancient Ideas to Artificial Intellgience*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.
5. Yoram Hirshfeld and Alexander Moshe Rabinovich. Logics for real time: Decidability and complexity. *Fundam. Inform.*, 62(1):1–28, 2004.
6. Z. Chaochen and M. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. EATCS Series of Monographs in Theoretical Computer Science. Springer, 2004.
7. V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal and duration calculi. *Journal of Applied Non-Classical Logics*, 14, 2004.
8. E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*. North-Holland Pub. Co., 1995.
9. Y. Venema. *Temporal Logic*. Blackwell Guide to Philosophical Logic, Blacwell Publishers, 1998.
10. J. L. Fiaderio and T. Maibaum. Action refinement in a temporal logic of objects. In *Temporal Logic*. LNCS, 1994.
11. R. L. Schwartz and P. M. Melliar-Smith. From state machines to temporal logic: Specification methods for protocol standards. *IEEE Trans. Commun. 30*, pages 2486–2496, 1982.
12. R. L. Schwartz, P. M. Melliar-Smith, and F. H. Voght. An interval logic for higher-level temporal reasoning. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 173–186. ACM Press, 1983.
13. B. Moszkowski. *Reasoning about Digital Circuits*. PhD thesis, Computer Science Department, Stanford University, 1983.
14. P. Ladkin. Logical time pieces. *AI Expert*, 2(8):58–67, 1987.
15. P. M. Melliar-Smith. Extending interval logic to real time systems. In *Proceedings of the Conference on Temporal Logic Specification*, pages 224–242. Springer, 1987.
16. R. Razouk and M. Gorlick. Real-time interval logic for reasoning about executions of real-time programs. *SIGSOFT Software Engineering Notes 14*, 8:10–19, 1989.
17. J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.
18. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.
19. Y. Venema. A modal logic for choppping intervals. *Journal of Logic and Computation*, 1:453–476, 1991.
20. J. F. van Benthem. *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse*. Kluwer, second edition, 1991.
21. A. Montanari, G. Sciavicco, and N. Vitacolonna. Decidability of interval temporal logics over split-frames via granularity. In *Proceedings of the 8th Europian Conference on Logics in AI*, pages 259–270. Springer, 2002.
22. N. Vitacolonna. *Intervals: Logics, Algorithms and Games*. PhD thesis, Department of Mathematics and Computer Science, University of Udine, 2005.
23. A. G. Walker. Durees et instants. *La Revue Scientifique*, (3266), 1947.
24. C. L. Hamblin. Instants and intervals. *Stadium Generale*, 27:127–134, 1971.
25. L. Humberstone. Interval semantics for tense logic: Some remarks. *Journal of Philosophical Logic*, 8:171–196, 1979.
26. D. Dowty. *Word Meaning and Montague Grammar*. Dordrecht: D. Reidel, 1979.
27. H. Kamp. Events, instants and temporal reference. In *Semantics from Different Points of View*, pages 376–417. Springer, 1979.
28. P. Röper. Intervals and tenses. *Journal of Philosophical Logic*, pages 451–469, 1980.
29. J. P. Burgess. Axioms for tense logic 2: Time periods. *Notre Dame Journal of Formal Logic*, 23(2):375–383, 1982.
30. J. F. van Benthem. *The Logic of Time*. Kluwer Academic Publishers, Dordrecht, 1983.
31. A. Galton. *The Logic of Aspect*. Claredon Press, Oxford, 1984.
32. P. Simons. *Parts, A Study in Ontology*. Claredon Press, Oxford, 1987.
33. J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.
34. J. F. Allen and J. P. Hayes. Moments and points in an interval-based temporal logic. In *Computational Intelligence*, pages 225–238. Blackwell Publishers, 1989.
35. J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–57, 1994.
36. A. Galton. A critical examination of Allen's theory of action and time. *Artificial Intelligence*, 42:159–198, 1990.
37. Grigore Rosu and Saddek Bensalem. Allen linear (interval) temporal logic Ð translation to LTL and monitor synthesis. In *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 263–277. Springer Berlin / Heidelberg, 2006.
38. R. Parikh. A decidability result for second order process logic. In *Proceedings of 19th FOCS*, pages 177–183. IEEE Computer Society Press, 1978.
39. V. R. Pratt. Process logic. In *Proceedings of 6th POPL*, pages 93–100. ACM, 1979.
40. D. Harel, A. Pnueli, and Y. Stavi. Process logic: Expressiveness, de-

cidability, completeness. *Journal of Computer and System Sciences*, 25:145–180, 1983.

41. J. Y. Halpern, Z. Manna, and B. Moszkowski. A high-level semantics based on interval logic. In *Proceedings of 10th ICALP*, pages 274–291, 1983.
42. D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 1. Clarendon Press, Oxford, 1994.
43. A. N. Prior. *Time and Modality*. Clarendon Press, Oxford, 1957.
44. A. N. Prior. *Past, Present and Future*. Oxford University Press, 1967.
45. A. N. Prior. *Papers on Time and Tense*. Oxford University Press, 1968.
46. J. A. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.
47. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.
48. A. Szalas. Temporal logic of programs: A standard approach. In *Time and Logic: A Computational Approach*, pages 1–50. UCL Press, 1995.
49. D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. *Conference Record of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.
50. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32:733–749, 1985.
51. M. Fisher. A resolution method for temporal logic. In *Proceedings of Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, 1991.
52. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM on Transactions of Computational Logic*, 2(1):12–56, 2001.
53. O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218. Springer-Verlag, 1985.
54. O. Lichtenstein and A. Pnueli. Propositional temporal logics: Decidability and completeness. *Logic Journal of the IGPL*, 8(1):55–85, 2000.
55. Mark Reynolds. The complexity of the temporal logic with "until" over general linear time. *J. Comput. Syst. Sci.*, 66(2):393–426, 2003.
56. Carsten Lutz, Dirk Walther, and Frank Wolter. Quantitative temporal logics over the reals: PSPACE and below. *Inf. Comput.*, 205(1):99–123, 2007.
57. Mark Reynolds. The complexity of temporal logic over the reals. *Annals of Pure and Applied Logic*, 161(8):1063–1096, 2010.
58. D. McDermott. A temporal logic for reasoning about process and plans. *Cognitive Science*, 6:101–155, 1982.
59. A. Rao and M. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of IJCAI*, 1993.
60. K. Abrahamson. *Decidability and Expressiveness of Logics of Programs*. PhD thesis, University of Washington, 1980.
61. M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 164–176. ACM, 1981.
62. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71. Springer-Verlag, 1982.
63. E. A. Emerson and J. Halpern. 'Sometimes' and 'Not Never' revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.
64. F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995.
65. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
66. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 169–180. ACM, 1982.
67. E. Emerson and Jai Srinivasan. Branching time temporal logic. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 123–172. Springer Berlin / Heidelberg, 1989.
68. E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.
69. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic. *ACM Transactions on Programming Languages and Systems*, 2(8):244–263, 1986.
70. Wojciech Penczek. Branching time and partial order in temporal logics. In *Time and Logic: A Computational Approach*, pages 179–228. UCL Press, 1995.
71. E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal of Compututation*, 29(1):132–158, 2000.
72. M. Reynolds. An axiomatization of full computation tree logic. *Journal of Symbolic Logic*, 66:1011–1057, 2001.
73. M. Reynolds. An axiomatization of PCTL. *Information and Computation*, 201(1):72–119, 2005.
74. François Laroussine and Philippe Schnoebelen. Specification in CTL+Past, verification in CTL. *Theor. Comput. Sci.*, 230(1-2):262–, 1999.
75. Laura Bozzelli. The complexity of CTL* + Linear Past. In Roberto Amadio, editor, *Foundations of Software Science and Computational Structures*, volume 4962 of *Lecture Notes in Computer Science*, pages 186–200. Springer Berlin / Heidelberg, 2008.
76. V. R. Pratt. On the composition of processes. In *Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '82, pages 213–223. ACM, 1982.
77. S. S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations (extended abstract). In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 28–37. ACM, 1984.
78. Y. Kornatzky and S. Pinter. An extension to partial order temporal logic (potl). Research Report 596, Department of Electrical Engi-

neering, Technion-Israel Institute of Technology, 1986.

79. Girish Bhat and Doron Peled. Adding partial orders to linear temporal logic. In *CONCUR '97: Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 119–134. Springer Berlin, 1997.
80. Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150(2):132 – 152, 1999.
81. Adrianna Alexander and Wolfgang Reisig. Compositional temporal logic based on partial order. In *Proceedings of the 11th International Symposium on Temporal Representation and Reasoning*, TIME '04, pages 125–132. IEEE Computer Society, 2004.
82. Alessio Lomuscio, Wojciech Penczek, and Hongyang Qu. Partial order reductions for model checking temporal epistemic logics over interleaved multi-agent systems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 659–666. International Foundation for Autonomous Agents and Multiagent Systems, 2010.
83. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT Press, 1996.
84. Jan Chomicki and David Toman. Temporal logic in information systems. In *Logics for Databases and Information Systems*, pages 31–70. Kluwer, 1998.
85. M. Abadi. The power of temporal proofs. *Theoretical Computer Science*, 65(1):35–83, 1989.
86. H. Andréka, I. Németi, and I. Sain. Mathematical foundations of computer science. *Lecture Notes in Computer Science*, pages 208–218, 1979.
87. Mark Reynolds. Axiomatising first-order temporal logic: until and since over linear time. *Studia Logica*, 57(2/3):279–302, 1996.
88. S. Merz. Decidability and incompleteness results for first-order temporal logics of linear time. *Journal of Applied Non-classical Logic*, pages 139–156, 1992.
89. J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Transactions on Database Systems*, 20:149–186, 1995.
90. R. Pliuskevicius. On the completeness and decidability of a restricted first order linear temporal logic. In *Proceedings of the 5th Kurt Gödel Colloquium on Computational Logic and Proof Theory*, pages 241–254. Springer-Verlag, 1997.
91. F. Wolter and M. Zakharyaschev. Axiomatizing the monodic fragment of first-order temporal logic. *Annals of Pure and Applied Logic*, pages 133–145, 2002.
92. H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, pages 217–274, 1998.
93. E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.
94. I. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic*, pages 85–134, 2000.
95. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.
96. Ian M. Hodkinson, Frank Wolter, and Michael Zakharyaschev. Monodic fragments of first-order temporal logics: 2000-2001 A.D. In *Proceedings of the Artificial Intelligence on Logic for Programming*, LPAR '01, pages 1–23. Springer-Verlag, 2001.
97. D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-Dimensional Modal Logics: Theory and Applications*. Elsevier, 2002.
98. A. Degtyarev, M. Fisher, and A. Lisitsa. Equality and monodic first-order temporal logic. *Studia Logica*, 72(2):147–156, 2002.
99. F. Wolter and M. Zakharyaschev. Modal description logics: Modalizing roles. *Fundamenta Informaticae*, pages 411–438, 1999.
100. C. Lutz, H. Sturm, F. Wolter, and M. Zakharyaschev. A tableau decision algorithm for modalized $\mathcal{ALC}$ with constant domains. *Studia Logica*, 72(2):199–232, 2002.
101. C. Dixon, M. Fisher, B. Konev, and A. Lisitsa. Practical first-order temporal reasoning. In *15th International Symposium on Temporal Representation and Reasoning, TIME '08*, pages 156–163. IEEE Computer Society Press, 2008.
102. E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinisvan. Quantitative temporal reasoning. In E. M. Clarke, A. Pnueli, and J. Sifakis, editors, *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*. LNCS, 1989.
103. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2:255–299, 1990.
104. Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–203, 1994.
105. T. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems*. PhD thesis, Department of Computer Science, Stanford University, 1991.
106. E. Allen Emerson and Richard J. Trefler. Generalized quantitative temporal reasoning: An automata theoretic-approach. In *In Proc. 7th Int. Joint Conf. Theory and Practice of Software Development (TAPSOFT'97*, pages 189–200. Springer, 1996.
107. François Laroussinie, Antoine Meyer, and Eudes Petonnet. Counting LTL. In *TIME 2010*, 2010.
108. Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS '08, pages 1–13. Springer-Verlag, 2008.
109. J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Proceedings 20th Annual IEEE Symposium on Logic in Computer Science*, pages 188–197. IEEE Computer Society Press, 2005.
110. Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
111. Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *ICALP*, pages 580–591, 1998.
112. Thomas A. Henzinger. It's about time: Real-time logics reviewed. In *CONCUR*, pages 439–454, 1998.
113. S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 9(2):1–27, 2008.

114. O. Maler, D. Nickovic, and A. Pnueli. Real time temporal logic: Past, present, future. In *Formal Modeling and Analysis of Timed Systems*, pages 2–16. Springer-Verlag, 2005.
115. Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. On expressiveness and complexity in real-time model checking. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 124–135. Springer-Verlag, 2008.
116. Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *FSTTCS*, pages 432–443, 2005.
117. Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 74–106. Springer-Verlag, 1992.
118. R. Alur, C. Courcoubetis, and D. L. Dill. Model checking for real-time systems. In *Proceedings 5th Conference on Logic in Computer Science*, pages 12–21. IEEE Computer Society Press, 1990.
119. F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In *CONCUR 2004 Ð Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer Berlin / Heidelberg, 2004.
120. H. A. Hansson. *Time and Probability in Formal Design and Distributed Systems*. PhD thesis, Department of Computer Science, Uppsala University, Sweden, 1991.
121. Daniele Beauquier and Anatol Slissenko. Polytime model checking for timed probabilistic computation tree logic. *Acta Informatica*, 35:645–664, 1998.
122. François Laroussinie, Antoine Meyer, and Eudes Petonnet. Counting CTL. In *FOSSACS*, pages 206–220, 2010.
123. F. Jahanian and A. K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering SE-12*, 9:890–904, 1986.
124. Farnam Jahanian and Aloysius K. Mok. Modechart: A specification language for real-time systems. *IEEE Trans. Softw. Eng.*, 20(12):933–947, 1994.
125. F. Jahanian and D. Stuart. A method for verifying properties of modechart specifications. In *Proceedings 9th Real-time Systems Symposium*, pages 12–21. IEEE Computer Society Press, 1988.
126. Stefan Andrei and Albert Mo Kim Cheng. Faster verification of RTL-specified systems via decomposition and constraint extension. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, pages 67–76. IEEE Computer Society, 2006.
127. Stefan Andrei and Albert M. K. Cheng. Verifying linear real-time logic specifications. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 333–342. IEEE Computer Society, 2007.
128. J. S. Ostroff and W. Wonham. Modeling and verifying real-time embedded computer systems. In *Proceedings of the 8th IEEE Real-Time Systems Symposium*, pages 124–132. IEEE Computer Society Press, 1987.
129. J. S. Ostroff. *Temporal Logic for Real-Time Systems*. Advanced Software Development Series. Research Studies Press Limited, 1989.
130. E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 402–413. IEEE Computer Society Press, 1990.
131. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and M. Trachtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16:403–414, 1990.
132. C. Ghezzi, D. Mandrioli, and A. Morzenti. TRIO, a logic language for executable specifications of real-time systems. *Journal of Systems and Software*, pages 107–123, 1990.
133. R. Mattolini. *TILCO: A Temporal Logic for the Specification of Real-Time Systems (TILCO: una Logica Temporale per la Specifica di Sistemi di Tempo Reale)*. PhD thesis, University of Florence, 1996.
134. R. Mattolini and P. Nesi. Using tilco for specifying real-time systems. In *ICECCS*, pages 18–, 1996.
135. R. Mattolini and P. Nesi. An interval logic for real-time system specification. *Software Engineering, IEEE Transactions on*, 27(3):208 –227, 2001.
136. L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS, 1994.
137. Pierfrancesco Bellini, Andrea Giotti, Paolo Nesi, and Davide Rogai. TILCO temporal logic for real-time systems implementation in C++. In *SEKE*, pages 166–173, 2003.
138. Pierfrancesco Bellini, Andrea Giotti, and Paolo Nesi. Execution of TILCO temporal logic specifications. In *Proceedings of the Eighth International Conference on Engineering of Complex Computer Systems*, ICECCS '02, pages 78–. IEEE Computer Society, 2002.
139. P. Bellini, P. Nesi, and D. Rogai. Reply to comments on "an interval logic for real-time system specification". *Software Engineering, IEEE Transactions on*, 32(6):428 –431, 2006.
140. P. Bellini, P. Nesi, and D. Rogai. Validating component integration with C-TILCO. *Electron. Notes Theor. Comput. Sci.*, 116:241–252, 2005.
141. M. Marx and M. Reynolds. Undecidability of compass logic. *Journal of Logic and Computation*, 9(6):897–914, 1999.
142. D. Marx and Y. Venema. *Multi-Dimensional Modal Logics*. Kluwer Academic Press, 1997.
143. K. Lodaya. Sharpening the undecidability of interval temporal logic. In *Proceeding of 6th Asian Computing Science Conference*, pages 290–298. LNCS, 2000.
144. Davide Bresolin, Dario Monica, Valentin Goranko, Angelo Montanari, and Guido Sciavicco. Decidable and undecidable fragments of halpern and shoham's interval temporal logic: Towards a complete classification. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR '08, pages 590–604. Springer-Verlag, 2008.
145. D. Bresolin. *Proof Methods for Interval Temporal Logics*. PhD thesis, University of Udine, 2007.
146. V. Goranko, A. Montanari, and G. Sciavicco. A general tableau method for propositional interval temporal logics. In *Proceedings of the International Conference Tableaux 2003*, pages 102–116. LNAI, 2003.
147. Ian Hodkinson, Angelo Montanari, and Guido Sciavicco. Non-finite

axiomatizability and undecidability of interval temporal logics with C, D, and T. In *CSL*, pages 308–322, 2008.

148. Z. Chaochen and M. Hansen. An adequate first order interval logic. In *Compositionality: the Significant Difference*, pages 584–608. LNCS, 1998.
149. V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighbourhood temporal logics. *Journal of Universal Computer Science*, 9(9):1137–1167, 2003.
150. D. Bresolin, A. Montanari, and P. Sala. An optimal tableau-based decision algorithm for propositional neighbourhood logic. In *Proceedings of STACS 2007: 24th International Symposium on Theoretical Aspects of Computer Science*, pages 549–560. LNCS, 2007.
151. D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco. On decidability and expressiveness of propositional interval neighbourhood logics. In *Proceedings of the International Symposium on Logical Foundations of Computer Science*, pages 84–99. LNCS, 2007.
152. D. Bresolin and A. Montanari. A tableau-based decision procedure for right propositional neighbourhood logic. In *Proceedings of TABLEAUX 2005*, pages 63–77. LNAI, 2005.
153. D. Bresolin, A. Montanari, and G. Sciavicco. An optimal tableau-based decision procedure for right propositional neighbourhood logic. *Journal of Automated Reasoning*, 38:173–199, 2007.
154. Davide Bresolin, Dario Della Monica, Valentin Goranko, Angelo Montanari, and Guido Sciavicco. Metric propositional neighborhood logics: Expressiveness, decidability, and undecidability. In *ECAI*, pages 695–700, 2010.
155. Davide Bresolin, Valentin Goranko, Angelo Montanari, and Guido Sciavicco. Propositional interval neighborhood logics: Expressiveness, decidability, and undecidable extensions. *Annals of Pure and Applied Logic*, 161(3):289 – 304, 2009.
156. A. V. Chagrov and M. N. Rybakov. How many variables does one need to prove PSPACE-hardness of modal logics? In *Advances in Modal Logic*, pages 71–82. King's College Publications, 2003.
157. S. Demri and R. Gore. An $o((n.logn)^3)$- time transformation from Grz into decidable fragments of classical first-order logic. In *Automated Deduction in Classical and Non-Classical Logics*, pages 153–167. LNAI, 2000.
158. Davide Bresolin, Valentin Goranko, Angelo Montanari, and Pietro Sala. Tableaux for logics of subinterval structures over dense orderings. *J. Log. Comput.*, 20(1):133–166, 2010.
159. Angelo Montanari, Ian Pratt-Hartmann, and Pietro Sala. Decidability of the logics of the reflexive sub-interval and super-interval relations over finite linear orders. In *TIME 2010*, 2010.
160. Jerzy Marcinkowski and Jakub Michaliszyn. The ultimate undecidability result for the Halpern-Shoham logic. In *LICS*, pages 377–386, 2011.
161. Ian Pratt-Hartmann. Temporal prepositions and their logic. *Artif. Intell.*, 166(1-2):1–36, 2005.
162. S. Konur. A decidable temporal logic for events and states. In *Thirteenth International Symposium on Temporal Representation and Reasoning (TIME 2006)*, pages 36–41. IEEE Computer Society Press, 2006.
163. Z. Chaochen, C. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.
164. B. Dutertre. On first-order interval temporal logic. Technical Report CSD-TR-94-3, Department of Computer Science, Royal Holloway College, University of London, 1995.
165. B. Moszkowski. A complete axiomatization of interval temporal logic with infinite time. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, pages 242–251. IEEE Computer Society Press, 2000.
166. D. P. Guelev. A complete proof system for first order interval temporal logic with projection. Technical Report 202, UNU/IIST, 2000.
167. Ben C. Moszkowski. Some very compositional temporal properties. In *PROCOMET*, pages 307–326, 1994.
168. Z. Chaochen, M. Hansen, and P. Sestoft. Decidability and undecidability results for duration calculus. In *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science*, pages 58–68. LNCS, 1993.
169. A. Rabinovich. Non-elementary lower bound for propositional duration calculus. *Information Processing Letters*, 66:7–11, 1998.
170. M. Fraenzle. Synthesizing controllers from duration calculus. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 168–187. LNCS, 1996.
171. D. Guelev and V. H. Dang. On the completeness and decidability duration calculus with iteration. In *Advances in Computer Science*, pages 139–150. LNCS, 1999.
172. N. Chetcuti-Serandio and L. Del Cerro. A mixed decision method for duration calculus. *Journal of Logic and Computation*, 10(6):877–895, 2000.
173. P. K. Pandya. Specifying and deciding quantified discrete-time duration calculus formulas using DCVALID. In *Real-Time Tools*, 2001.
174. C. Zhou. Linear duration invariants. In *Proceedings of the Third International Symposium Organized Jointly with the Working Group Provably Correct Systems on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 86–109. Springer-Verlag, 1994.
175. L. X. Dong and D. V. Hung. Checking linear duration invariants by linear programming. In *Concurrency and Parallelism, Programming, Networking, and Security*, pages 321–332. Springer-Verlag, 1996.
176. P. H. Thai and D. V. Hung. Checking a regular class of duration calculus models for linear duration invariants. In *Proceedings of the International Symposium on Software Engineering for Parallel nd Distributed Systems*, pages 61–71. IEEE Computer Society Press, 1998.
177. P. H. Thai and D. V. Hung. Verifying linear duration constraints of timed automata. In *Proceedings of ICTAC'04*, pages 295–309. Springer-Verlag, 2004.
178. M. Satpathy, D. V. Hung, and P. K. Pandya. Some decidability results for duration calculus under synchronous interpretation. In *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 186–197. Springer-Verlag, 1998.
179. M. Fränzle. Take it NP-Easy: Bounded model construction for duration calculus. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 245–264. Springer-Verlag, 2002.

180. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207. Springer-Verlag, 1999.
181. Martin Fränzle and Michael R. Hansen. Deciding an interval logic with accumulated durations. In *TACAS*, pages 201–215, 2007.
182. M. R. Hansen and R. Sharp. Using interval logics for temporal analysis of security protocols. In *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering*. ACM, 2003.
183. R. Barua, S. Roy, and Z. Chaochen. Completeness of neighbourhood logic. *Journal of Logic and Computation*, 10(2):271–295, 2000.
184. R. Barua and Z. Chaochen. Neighbourhood logics. Research Report 120, UNU/IIST, 1997.
185. R. Alur and D.L. Dill. Automata-theoretic verification of real-time systems. *Formal Methods for Real-Time Computing*, pages 55–82, 1996.
186. R. Alur, T. A. Henzinger, and P. H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.
187. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - A tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON Workshop on Hybrid systems III : Verification and Control*, pages 232–243. Springer-Verlag, 1996.
188. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 546–550. Springer-Verlag, 1998.
189. P. K. Pandya. Interval duration logic: Expressiveness and decidability. In *Proceedings of TPTS*, 2002.
190. B. Sharma, P. Paritosh, and C. Supratik. Bounded validity checking of interval duration logic. In *TACAS 2005: Tools and Algorithms for the Construction and Analysis of Systems*, pages 301–316. Springer-Verlag, 2005.
191. G. Chakravorty and P. Pandya. Digitizing interval duration logic. In *Computer Aided Verification*, pages 167–179. Springer-Verlag, 2003.
192. J. C. Filliatre, S. Owre, H. Ruess, and N. Shanka. ICS: Integrated canonizer and solver. In *Computer Aided Verification*, pages 246–249. Springer-Verlag, 2002.
193. D. V. Hung and Z. Chaochen. Probabilistic duration calculus for continuous time. In *Formal Aspects of Computing*, pages 21–44, 1994.
194. R. Fagin and J. Y. Halpern. Reasoning about knowledge and probability. *Journal of the ACM*, 41(2):340–367, 1994.
195. Z. Marković, Z. Ognjanović, and M. Rasković. A probabilistic extension of intuitionistic logic. *Mathematical Logic Quarterly*, 49:415–424, 2003.
196. H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proceedigns of Real-time Systems Symposium*, pages 102–111. IEEE, 1989.
197. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:102–111, 1994.
198. Tomás Brázdil, Vojtech Forejt, Jan Kretínský, and Antonín Kucera. The satisfiability problem for Probabilistic CTL. In *LICS*, pages 391–402, 2008.
199. A. Aziz, V. Singhal, and F. Balarin. It usually works: The temporal logic of stochastic systems. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 155–165. Springer-Verlag, 1995.
200. A. Bianco and L. D. Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer-Verlag, 1995.
201. Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. In *Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *Lecture Notes in Computer Science*, pages 75–95. Springer Berlin / Heidelberg, 1999.
202. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *In Proc. 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2000.
203. M. Jurdzinski, F. Laroussinie, and J. Sproston. Model checking probabilistic timed automata with one or two clocks. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 170–184. Springer Berlin / Heidelberg, 2007.
204. Z. Ognjanović. Discrete linear-time probabilistic logics: Completeness, decidability and complexity. *Journal of Logic and Computation*, 16(2):257–285, 2006.
205. Z. Liu, A. P. Ravn, E. V. Sorensen, and C. Zhou. A probabilistic duration calculus. Technical report, University of Warwick, 1992.
206. D. V. Hung and M. Zhang. On verification of probabilistic timed automata against probabilistic duration properties. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 165–172. IEEE Computer Society, 2007.
207. Dang Van Hung Choe Changil. Model checking durational probabilistic systems against probabilistic linear duration invariants. Research Report 337, UNU/IIST, 2006.
208. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.
209. D. P. Guelev. Probabilistic neighbourhood logic. In *Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 264–275. Springer-Verlag, 2000.
210. Dimitar Guelev. Probabilistic neighbourhood logic. Research Report 196, UNU/IIST, 2000.
211. D. P. Guelev. Probabilistic interval temporal logic. Technical Report 144, UNU/IIST, 1998.
212. Krister Segerberg. A completeness theorem in the modal logic of programs. In T. Traczyk, editor, *Universal Algebra*, volume 9, pages 31–46. Banach Centre Publications, 1982.
213. Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

214. David Harel, Jerzy Tiuryn, and Dexter Kozen. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000.
215. Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.
216. L Giordano, A Martelli, and C Schwind. Reasoning about actions in dynamic linear time temporal logic. *Logic Journal of IGPL*, 9(2):273–288, 2001.
217. R Kowalski and M Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.
218. Y Shoham. *Reasoning about Action and Change*. MIT Press, 1987.
219. Raymond Reiter. Proving properties of states in the situation calculus. *Artif. Intell.*, 64(2):337–351, 1993.
220. Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *The Journal of Logic Programming*, 17(2,3,4):301 – 321, 1993.
221. Nuel D. Belnap. *Facing the Future: Agents and Choices in Our Indeterminist World*. Oxford University Press, USA, 2001.
222. D. Kozen. Semantics of probabilistic programs. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 101–114. IEEE Computer Society, 1979.
223. Y. A. Feldman and D. Harel. A probabilistic dynamic logic. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 181–195. ACM, 1982.
224. V. R. Pratt. Semantical considerationsn on Floyd-Hoare logic. Technical report, MIT, 1976.
225. Y. A. Feidman. A decidable propositional probabilistic dynamic logic. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 298–309. ACM, 1983.
226. D. Kozen. A probabilistic PDL. In *Proceedings of the fifteenth annual ACM symposium on Theory of Computing*, pages 291–297. ACM, 1983.
227. Y. A. Feldman. A decidable propositional dynamic logic with explicit probabilities. *Information Control*, 63(1-2):11–38, 1984.
228. K. Segerberg. Qualitative probability in a modal setting. *Proceedings of the Second Scandinavian Logic Symposium*, pages 575–604, 1971.
229. D. P. Guelev. A propositional dynamic logic with qualitative probabilities. *Journal of Philosophical Logic*, 28:575–604, 1999.
230. R. Cleaveland, S. P. Iyer, and M. Narasimha. Probabilistic temporal logics via the modal Mu-Calculus. *Theoretical Computer Science*, 342(2-3):316–350, 2005.
231. S. Konur. *An Interval Temporal Logic for Real-time System Specification*. PhD thesis, Department of Computer Science, University of Manchester, UK, 2008.

Savas Konur is a member of staff at the Department of Computer Science, University of Liverpool. He obtained a Ph.D. degree in Computer Science from University of Manchester in 2008. He is currently a member of Logic and Computation research group in the Department of Computer Science, University of Liverpool. His research interests include temporal reasoning, formal specification and verification, model checking, real-time systems, multi-agent systems and pervasive systems.