Theoretical Computer Science 503 (2013) 61–88

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Combined model checking for temporal, probabilistic, and real-time logics

Savas Konur *, Michael Fisher, Sven Schewe

Department of Computer Science, University of Liverpool, Liverpool, UK

ARTICLE INFO

Article history:
Received 30 June 2012
Received in revised form 12 February 2013
Accepted 16 July 2013
Communicated by P. Aziz Abdulla

Keywords:
Formal verification
Model checking
Combination of logics
Complexity
Multi-agent systems

ABSTRACT

Model checking is a well-established technique for the formal verification of concurrent and distributed systems. In recent years, model checking has been extended and adapted for multi-agent systems, primarily to enable the formal analysis of belief–desire–intention systems. While this has been successful, there is a need for more complex logical frameworks in order to verify realistic multi-agent systems. In particular, probabilistic and real-time aspects, as well as knowledge, belief, goals, etc., are required.

However, the development of new model checking tools for complex combinations of logics is both difficult and time consuming. In this article, we show how model checkers for the constituent *temporal, probabilistic, and real-time logics* can be re-used in a modular way when we consider combined logics involving different dimensions. This avoids the re-implementation of model checking procedures. We define a modular approach, prove its correctness, establish its complexity, and show how it can be used to describe existing combined approaches and define yet-unimplemented combinations. We also demonstrate the feasibility of our approach on a case study.

1. Introduction

Model checking is a powerful approach for the formal verification of computer systems. In the area of verification, model checking has been studied extensively, and has become a well-established area of research and technology [1].

In recent years model checking techniques have been applied to the verification of multi-agent systems. Multi-agent systems comprise many different facets, which often makes their formal description hard. They are therefore often difficult to describe formally. The verification of multi-agent systems is also difficult, because there are often many different dimensions to look at simultaneously, including: autonomous behaviour of agents; beliefs, desires and intentions; teamwork; uncertainty in sensing and communication; real-time aspects; etc. Thus, we may want to represent not only the basic dynamic behaviour of a multi-agent system, but also several of the various aspects mentioned above. Modal logics can specify some of the concepts, such as knowledge, beliefs, intentions, norms, and temporal aspects. However, applying model checking techniques that were introduced for standard temporal logics, such as LTL [2] or CTL [3], to the verification of multi-agent systems [4] is not straightforward. In order to overcome this problem, researchers have attempted to extend basic model checking techniques by adding further operators. These extensions have been carried out based on sophisticated models of autonomous behaviour, uncertainty, and interaction [5].

* Corresponding author.
E-mail addresses: Konur@liverpool.ac.uk (S. Konur), MFisher@liverpool.ac.uk (M. Fisher), Sven.Schewe@liverpool.ac.uk (S. Schewe).

0304-3975
http://dx.doi.org/10.1016/j.tcs.2013.07.012

1.1. State of the art

Model checking temporal–epistemic aspects of multi-agent systems has recently received considerable attention. However, the large amount of individual aspects of multi-agent systems has resulted in an unmanageable set of their combinations in the literature.

In [6], for example, some theoretical analysis is done for the model checking problem of epistemic linear temporal logics on "interpreted systems with perfect recall". In [7] an approach to the model checking of a temporal logic of knowledge, CKL_n, which combines LTL with epistemic logic, was developed. With this approach, *local propositions* provide a means to reduce CKL_n model checking to LTL model checking. In [4] a bounded model checking technique is applied to the epistemic logic of branching time CTLK, which comprises both CTL and knowledge operators. Work in [8] describes the tool MCK for model checking the logic of knowledge, which supports a range of knowledge semantics and choices of temporal language. [9] studies using the model checker tool NuSMV for the verification of epistemic properties of multi-agent systems. The work [10] presents a model checking algorithm based on SMV for the logic CKK_n, which combines CTL and epistemic operators. The work in [11] presents the MCMAS tool, which was developed for verifying temporal and epistemic operators in interpreted systems. A multi-valued μK-calculus, which can specify knowledge and time in multi-agent systems, is presented in [12]. In [13], the authors introduce a symbolic model checking algorithm, based on "interpreted systems with local propositions" semantics, for the logic CKL_n via 'Ordered Binary Decision Diagrams (OBDDs)' and develop the MCTK model checker. The work reported in [14] presents a verification technique, based on the model checker NuSMV, for the logic CTLK. In [15] multi-agent systems are verified by means of a special action-based temporal logic, ACTLW. On the theoretical side, [16,17] study combinations of the epistemic modal logic $S5_n$ with temporal logics. Finally, Dennis et al. have developed a practical framework for model checking belief–desire–intention (BDI) properties of multi-agent programs [18].

Real-time and epistemic aspects of multi-agent systems have also been considered. The real-time temporal knowledge logic RTKL, which combines real-time and knowledge operators, is introduced in [19], and a model checking algorithm for this logic is presented. Furthermore, the authors extend RTKL with *cooperation* modalities and obtain the logic RATKL. Lomuscio et al. [20] present the real-time extension of the existential fragment of CTLK, and extend the corresponding bounded model checking algorithm based on a discretisation method.

The relations between knowledge and probability were also investigated in the domain of multi-agent systems [21,22]. Delgado et al. proposed an epistemic extension of the probabilistic CTL temporal logic, called KPCTL, allowing epistemic and temporal properties as well as likelihoods of events in [23], where the authors also describe how to extend the PRISM model checker [24] to verify KPCTL formulas over probabilistic multi-agent system models.

1.2. Contribution

The main drawback of the research mentioned above is that numerous combined logics have been introduced to represent different views on multi-agent systems, and this has required the implementation of many different verification systems.

In this article we present a different approach to model checking. Instead of introducing new logics for combinations of different aspects, analysing the model checking problem, and implementing a new checker for a particular combination, we combine logics representing different aspects, using a *generic* model checking method suitable for most of these different combinations of logics. In this way, many aspects of multi-agent systems, such as knowledge and time, knowledge and probability, real-time and knowledge, etc., can be represented as a combination of logics. A combined model checking procedure can then be synthesised from model checkers for constituent logics. The component logics we have in mind are logics that refer to key aspects of multi-agent systems, including:

- classic temporal logics (CTL, LTL, etc.);
- belief/knowledge logics (modal logics KD45, S5, etc.);
- logics of goals (modal logics KD, etc.);
- probabilistic temporal logics (PCTL, etc.); and
- real-time temporal logics (TCTL, etc.).

While the formal description of multi-agent systems is essentially multi-dimensional, we generally do not have verification tools for all appropriate combinations. For example, we might have separate verification tools for logics of knowledge, logics of time, real-time temporal logics, or probabilistic temporal logics, but we have no tool that can verify a description containing *all* these dimensions.

In this article we study results and methods for model checking of combined logics. This can be seen as an extension of the work of Franceschet et al. [25], where model checking for combinations of modal and (simple) temporal logics are considered. They study combinations of logics, whose semantics is defined as standard *Kripke Structures* $\langle W, R, l\rangle$, where W is the underlying set of worlds/states, l is a labelling function describing the propositions true at each world/state, and R is the accessibility/transition relation between elements of W. The main drawback of the work in [25] is that their framework does not capture more complex logics, whose semantics is not of the above form. Specifically, models of probabilistic temporal logics can also be represented in the $\langle W, R, l\rangle$ form, but here R is not just a simple relation between pairs of

states/worlds, but also includes a probability distribution for transitions. Similarly, real-time temporal logics have a relation R that is extended with a set of clock constraints, and the model itself is extended with a finite set of clocks. It may well be that some probabilistic/real-time temporal logics can be transformed into standard propositional temporal logics, allowing for the application of the techniques of Franceschet et al. [25], but it is likely that blow-up incurred in the structure will be prohibitive.

In this article, we extend the framework from [25] such that it can be used for more complex combinations of propositional logics. This allows us to combine *logics of time, logics of belief, logics of intentions, probabilistic temporal logics, and real-time temporal logics*, etc., in order to provide a coherent framework for the formal analysis of multi-agent systems. We provide a generic model checking algorithm, which synthesises a combined model checker from the model checkers of simpler component logics. We show that the method terminates, and is both sound and complete. We also analyse the computational complexity of the resulting method, and show that the complexity of the synthesised model checker is essentially the supremum of the complexities of the component model checkers. This result suggests that modularity is easier to obtain for model checking than for deductive approaches, where combination can lead to exponential (or worse) complexity.

We also show that our combination method is quite useful in determining model checking complexities of some existing logics. Using our generic approach, we (re-)prove the model checking complexities of several logics, such as the polynomial bound for the branching time epistemic logic, CTLK [26], the PSPACE bound for the linear time epistemic logic, CKL_n [7], polynomial bound for the probabilistic epistemic logic KPCTL [23], and the PSPACE bound for the real-time epistemic logic TECTLK [20].

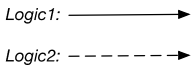
1.3. Organisation of the article

In Section 2, we provide a brief review of logical combination methods. In Section 3, we introduce the notation that will be used throughout the article. We then present the combined model checking algorithm for the modal case, and establish its completeness and complexity in Section 4. In Section 5, we extend the combined model checking algorithm to include real-time logics, apply our method to some existing logics, and present some complexity results. Finally, in Section 6, we apply the combination method to the *Scatterbox* message-forwarding system to demonstrate the usefulness of our generic model checking approach. We close with a discussion of our results and potential future work in Section 7.

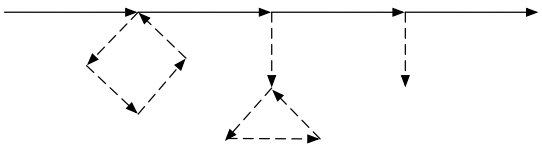
2. Combinations of logics

Much work has been carried out on the combination of various modal/temporal logics. *Temporalisation*, *fusion* (*or independent combination*), and *product* (*or join*) are among the popular forms of logic combination [27–30].

To provide an overview of these different types of combinations, let us first assume that we want to combine two logics, *Logic1* and *Logic2*. Further, let us assume that *Logic1* is a temporal logic of some form. Let us represent these graphically as

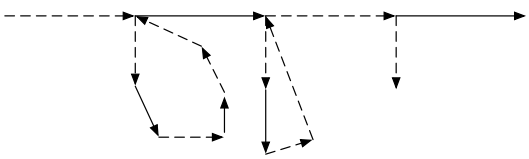


Temporalisation. This is a method that adds a temporal dimension to another logic system. With this method, an arbitrary logic system is combined with temporal modalities to create a new system. Basically a combination of two logics, A and B, resulting in a logic A(B), where a pure subformula of B can be treated as an atom within A. Consequently, the combination is not *symmetric* — the logic A is the main one, but at each world/state described by A, we might have a formula of B that describes a 'B-world'.



Temporalisation is investigated in [31], where the logical properties soundness, completeness, and decidability are analysed. In [25] a model checking procedure is presented for temporalised logics. However, the procedure only covers combining modal propositional temporal logics.

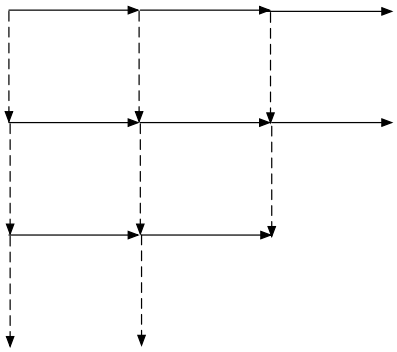
Fusion (or 'Independent Join'). The independent combination of two logic systems put together the expressive power of the two component logic systems in an *unrestricted* way [25]. In other words, independent combination of two logics over a set of atomic propositions is obtained by the union of the respective sets of connectives and the union of the formation rules of both logics [31]. Unlike temporalisation, the fusion of two logics A and B, denoted $\text{A} \oplus \text{B}$, is symmetric. That is, at any state/world we can either take an 'A-step' or a 'B-step'.



Note that the two logics are essentially independent (hence the name). This means properties such as $OP_A OP_B \varphi \Leftrightarrow OP_B OP_A \varphi$, where OP_X is an operator in logic X, are rarely valid. (For tighter linking of this kind we use the product combination below.) The logical properties of fusion, such as soundness, completeness, and decidability, are analysed in [31], while [25] tackles the model checking problem for the fusion of modal temporal logics.

Product (or 'Join'). In both, temporalisation and independent join, formulas are evaluated at a single node of a model. In contrast to this, the join of two temporal system flows of times are considered over a higher-dimensional plane. Thus, when using the join method, it is possible to produce higher-dimensional temporal logics by combining lower-dimensional temporal logics.

This product combination is similar to fusion, but with a much tighter integration of the logics. The join of two logics A and B is denoted $A \otimes B$, and can be visualised as follows.



Specifically, the operators of the constituent logics tend to be commutative. That is, we have axioms such as $\vdash OP_A OP_B \varphi \Leftrightarrow OP_B OP_A \varphi$. The product construction is investigated in [31], where logical properties of soundness, completeness, and decidability are analysed. In [25] a combined model checking procedure for modal temporal logic products is presented.

The decision problem for product logics is typically more complex than that for fused logics, with products of relatively benign logics even becoming undecidable. For this reason, product logics are rarely used in practise, with fusions being the main tool for combining logics in a deductive way. However, we argue that this disadvantage does not extend to model checking product logics. This observation turns the model checking problem for products of logics into a feasible approach.

We end this section by giving some results for the combining methods mentioned above. Most of the work carried out on combining logics has been from a *deducibility* or *expressiveness* point of view. To generalise many years of important research, we can say that, usually, decidability and axiomatisability properties transfer to both temporalisations and fusions from the constituent logics [32,33]. Thus, the fusion of two decidable 'normal polyadic polymodal logics' is decidable [34]. Work in [35] obtained decidability for modal logics with the 'converse operator interpreted over transitive frames'. The temporalisation and fusion of the one-dimensional logic PLTL, i.e., PLTL(PLTL) and $\text{PLTL} \oplus \text{PLTL}$, respectively, are also found to be decidable [31,27]. In the case of join, the combinations are less tame: $\text{S5} \otimes \text{S5}$ is NEXPTIME-complete [36], S5^3 is undecidable [37], and $\text{PLTL} \otimes \text{PLTL}$ is undecidable [38], too.

3. Preliminaries

In this section, we define the notation that will be used throughout the article. The logics we will define are mainly branching logics, because most of the examples we will discuss later are based on branching-time temporal logics.

3.1. Computation tree logic (CTL)

CTL is a branching-time temporal logic with a discrete notion of time and only future modalities. The temporal operators of CTL allow us to express properties about *some* or *all* computations. CTL is 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. CTL formulas are interpreted over transition systems, which are a class of *Kripke structures*. The execution of a transition system constructs a set of *paths*, which are infinite sequences of states. The ith element of a path σ is denoted by $\sigma[i]$. The set of paths from a state s is denoted by $Paths(s)$.

Definition 3.1. Assume $\mathcal{M} = \langle S, \mathcal{E}, l\rangle$ is a transition system, where S is a finite set of *states*, $\mathcal{E}$ is a transition relation and $S \to 2^{AP}$ is a labelling function. For a given CTL-formula φ and state $s \in S$, the satisfaction relation $\models$ is inductively defined on the structure of φ as follows:

$$\mathcal{M}, s \models p \quad \text{iff} \quad p \in l(s)$$
$$\mathcal{M}, s \models \neg\varphi \quad \text{iff} \quad \mathcal{M}, s \not\models \varphi$$
$$\mathcal{M}, s \models \varphi_1 \wedge \psi_2 \quad \text{iff} \quad \mathcal{M}, s \models \varphi_1 \text{ and } \mathcal{M}, s \models \varphi_2$$
$$\mathcal{M}, s \models \exists\psi \quad \text{iff} \quad \mathcal{M}, \sigma \models \psi \text{ for some } \sigma \in Paths(s)$$
$$\mathcal{M}, s \models \forall\psi \quad \text{iff} \quad \mathcal{M}, \sigma \models \psi \text{ for all } \sigma \in Paths(s)$$

Path formulas are defined on the following semantics:

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

3.2. Probabilistic CTL (PCTL)

PCTL [39] is an extension of CTL, which can express statements such as "the probability that a process eventually terminates is 0.2", "the probability that the system energy exceeds a certain value is $\geqslant 0.8$", etc. PCTL is interpreted over (*Markov chains* or) *Markov decision processes*.

Definition 3.2. A *Markov decision process* is a tuple $\langle S, A, \mathcal{E}, \mu, l\rangle$, where

- S is a finite set of *states*;
- A is a finite set of *actions*;
- $\mathcal{E} \subseteq S \times A \times S$ is a finite set of *edges*;
- $\mu : \mathcal{E} \to [0, 1]$ is a *probability transition function*,[1] such that, for all $s \in S$ and $a \in A$, $\sum_{e \in \mathcal{E}_s^a} \mu(e) = 1$, where $\mathcal{E}_s^a$ denotes the edges from s with the action a chosen, i.e., $\mathcal{E}_s^a = \{\langle s, a, s'\rangle \in \mathcal{E}\}$;
- $l : S \to 2^{AP}$ is a labelling function.

The execution of a Markov decision process (MDP) constructs a set of *paths*, which are infinite sequences of states. A probability measure π_m for a set of paths with a common prefix of the length n, $s_0 \to s_1 \to \cdots \to s_n$, is defined to be the product of transition probabilities along the prefix, i.e., $\mu(\langle s_0, a_1, s_1\rangle) \times \cdots \times \mu(\langle s_{n-1}, a_n, s_n\rangle)$ [39].

Apart from the usual operators from classical logic such as $\wedge$ (and), $\vee$ (or) and $\Rightarrow$ (implies), PCTL has the *probabilistic operator* $\mathrm{P}_{\sim r}$, where $0 \leqslant r \leqslant 1$ is a *probability bound* and $\sim \in \{<, >, \leqslant, \geqslant, =\}$. Intuitively, a state s of a model satisfies $\mathrm{P}_{\sim r}[\varphi]$ if, and only if, the probability of taking a path from s that satisfies the *path formula* φ is bounded by '$\sim r$'. The following path formulas φ are allowed: $\bigcirc\varphi$, $\varphi\mathcal{U}\psi$ and $\varphi\mathcal{U}^{\leqslant k}\psi$ (where $k \in \mathbb{N}$). The semantics of PCTL formulas is given below:

Definition 3.3. Assume $\mathcal{M} = \langle S, A, \mathcal{E}, \mu, l\rangle$ is a Markov decision process. For a given PCTL-formula φ and a state $s \in S$, the satisfaction relation $\models$ is inductively defined on the structure of φ as follows [39]:

$$\mathcal{M}, s \models p \quad \text{iff} \quad p \in l(s)$$
$$\mathcal{M}, s \models \neg\varphi \quad \text{iff} \quad \mathcal{M}, s \not\models \varphi$$
$$\mathcal{M}, s \models \varphi_1 \wedge \varphi_2 \quad \text{iff} \quad \mathcal{M}, s \models \varphi_1 \text{ and } \mathcal{M}, s \models \varphi_2$$
$$\mathcal{M}, s \models \mathrm{P}_{\sim r}[\psi] \quad \text{iff} \quad \pi_m\big(\sigma \in Paths(s) \text{ s.t. } \mathcal{M}, \sigma \models \psi\big) \sim r$$

Path formulas are defined on the following semantics:

[1] In order to keep the syntax simple, we consider only one probability distribution μ. It is straightforward to modify our definition to accommodate a finite set of probability distributions $\prod = \{\mu_1, \ldots, \mu_n\}$.

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

As an example, the property that "the probability of φ eventually occurring is greater than or equal to b" can be expressed in PCTL as follows:

$$\mathrm{P}_{\geqslant b}[\mathtt{true}\,\mathcal{U}\varphi].$$

3.3. Timed automata

A timed automaton [40] is a transition system with a finite set of real-valued *clocks*. As time progresses, all clocks increase their values at the same rate. In each transition, some clocks can be reset, and the remaining clocks continue to proceed their values. Therefore, clock values denote the elapse of time units since the last reset of the clocks. In a timed automaton, transitions are associated with clock constraints, which are called *guards* (enabling conditions). A transition is enabled if the corresponding guard is satisfied by the current clock values. Assume C is a set of clocks. The set of clock constraints over C, denoted by $\Psi(C)$, is defined by the following syntax:

$$\psi ::= true \mid x \sim k \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

where $x \in C$, $k \in \mathbb{N}$ and $\sim \in \{<, >, \leqslant, \geqslant, =\}$. When a transition is enabled, a set of clocks are reset to zero, while the remaining clocks keep their values. In addition to clock constraints enabling transitions, timed automata might also contain clock constraints that are used to restrict the time spent in locations. Such constraints are called *invariances*. They are defined in a similar way to guards.

Definition 3.4. (See [40].) A *timed automaton* $\mathcal{A}$ is a tuple $\langle L, A, C, \mathcal{E}, inv, l\rangle$, where

- L is a finite set of locations;
- A is a finite set of actions;
- C is a finite set of clocks;
- $\mathcal{E} \subseteq L \times \Psi(C) \times A \times 2^C \times L$ is a set of edges between locations;
- $inv : L \to \Psi(C)$ is a function associating each location with some clock constraint in $\Psi(C)$;
- $l : L \to 2^{AP}$ is a labelling function (AP is a set of atomic propositions).

In the definition above, the invariance function inv assigns clock constraints to locations. The automaton might stay in a location ℓ while $inv(\ell)$ holds. As soon as $inv(\ell)$ becomes false, the automaton makes a transition, such that the current clock values satisfy the clock constraints (guard) of this transition. Let $\ell, \ell' \in L$ be two locations, $\gamma \in \Psi(C)$ be a clock constraint, $a \in A$ be an action, and $X \subseteq C$ be a set of clocks. An *edge* (*transition*) is denoted by a tuple $\langle \ell, \gamma, a, X, \ell'\rangle$, which can be informally interpreted as "the automaton moves from ℓ to ℓ' if the current clock values satisfy the clock constraint γ; if this is the case, the action a is taken and all the clocks of X are reset to zero".

Here, we define the semantics of timed automata. To do this, we first introduce further notation. Clock constraints are defined on clock valuations. A *clock valuation* ν over a clock set C is a mapping from C to $\mathbb{R}_{\geqslant 0}$, i.e., $\nu : C \to \mathbb{R}_{\geqslant 0}$. The value of a clock $x \in C$ is denoted by $\nu(x)$. The set of all clock valuations of C is denoted by $\mathbb{R}^C_{\geqslant 0}$. Let $\nu \in \mathbb{R}^C_{\geqslant 0}$ be a clock valuation and $d \in \mathbb{R}_+$ be a positive real number. $\nu + d$ denotes the clock valuation which assigns each clock $x \in C$ to the value $\nu(x) + d$. $\nu[x]$ denotes the clock valuation which resets x to 0, and keeps the values of other clocks same. Also, we write $\nu \models x \sim k$ if, and only if, ν satisfies $x \sim k$, i.e., $\nu(x) \sim k$.

We can define the semantics of a timed automaton in terms of a *timed transition system*:

Definition 3.5. (See [41].) A *timed transition system* $\mathcal{T}_\mathcal{A}$ of a timed automaton $\mathcal{A} = \langle L, A, C, \mathcal{E}, inv, l\rangle$ is a tuple $\langle S, A_\mathcal{T}, \to, l_\mathcal{T}\rangle$, where

- $S = \{\langle \ell, \nu\rangle \mid \ell \in L,\ \nu \in \mathbb{R}^C_{\geqslant 0} \text{ s.t. } \nu \models inv(\ell)\}$,
- $A_\mathcal{T} = A \cup \mathbb{R}_+$,
- $\to \subseteq S \times \{A \cup \mathbb{R}_+\} \times S$ can be defined in either of the rules below:
  - $\langle \ell, \nu\rangle \xrightarrow{d} \langle \ell, \nu + d\rangle$ iff $\exists d \in \mathbb{R}_+$ s.t. $\nu + d' \models inv(\ell)$, $\forall (0 \leqslant d' \leqslant d)$,
  - $\langle \ell, \nu\rangle \xrightarrow{a} \langle \ell', \nu'\rangle$ iff $\langle \ell, \gamma, a, X, \ell'\rangle \in \mathcal{E}$, $\nu \models \gamma$, $\nu' = \nu[X]$ and $\nu' \models inv(\ell')$,
- $l_\mathcal{T}(\langle \ell, \nu\rangle) = l(\ell) \cup \{\gamma \in \Psi(C) \mid \nu \models \gamma\}$.

A *configuration* of a timed automaton $\mathcal{A}$ is a state of the corresponding transition system $\mathcal{T}_\mathcal{A}$, which is a pair $\langle \ell, \nu\rangle$, where $\ell \in L$ is a location and $\nu \in \mathbb{R}^C_{\geqslant 0}$ is a valuation. $\mathcal{T}_\mathcal{A}$ has an infinite set of states. Each *path* (*execution*) σ of $\mathcal{T}_\mathcal{A}$ corresponds to

a possible behaviour of the automaton $\mathcal{A}$, and it is defined as an infinite sequence $s_0 \xrightarrow{\xi_0} s_1 \xrightarrow{\xi_1} s_2 \xrightarrow{\xi_2} \cdots$, where $\xi_i \in A \cup \mathbb{R}_+$ (for $i \in \mathbb{N}$). The ith element of a path is denoted by $\sigma[i]$. The set of all paths from a state s is denoted by $Paths(s)$. The elapsed time of reaching a state s' from a state s within a path σ is the sum of the delays along the path, and it is denoted as $\mathbb{T}(s \xrightarrow{\sigma} s')$.

As mentioned above, $\mathcal{T}_\mathcal{A}$ has an infinite number of states. However, some of the states do not differ for some clock valuations in the sense that they display a similar behaviour modulo bisimulation. We therefore define the notion of *clock equivalence*, which is used to obtain a finite region graph.

Let *integral* and *fractional* parts of $d \in \mathbb{R}_+$ be denoted by $\lfloor d \rfloor$ and $frac(d)$, respectively, and c_x be the largest constant for $x \in C$ that is compared in some clock constraint in $\mathcal{A}$ or in a formula φ. (Clock constraints might also exist in logical formulas, cf. Section 3.4.) Given two clock valuations ν and ν', ν is *equivalent* to ν', denoted $\nu \cong \nu'$, if, and only if, the following is true [40]:

- $\forall x \in C,\ \nu(x) > c_x$ iff $\nu'(x) > c_x$ and
- $\forall x, y \in C,\ \nu(x) \leqslant c_x$ and $\nu(y) \leqslant c_y$ implies
  - $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$,
  - $frac(\nu(x)) = 0$ iff $frac(\nu'(x)) = 0$, and
  - $frac(\nu(x)) \leqslant frac(\nu(y))$ iff $frac(\nu'(x)) \leqslant frac(\nu'(y))$.

Let $\cong$ be a clock equivalence on C. For any $\gamma \in \Psi(C)$, $\nu \cong \nu'$ implies that $\nu \models \gamma$ iff $\nu' \models \gamma$ [40]. The *equivalence class* of $\nu \in \mathbb{R}^C_{\geqslant 0}$, which is denoted by $[\nu]$, is defined by $[\nu] = \{\nu' \in \mathbb{R}^C_{\geqslant 0} \mid \nu \cong \nu'\}$. Since the set of equivalence classes is finite, this allows us to define a finite region graph.

Definition 3.6. (See [41].) Given a timed automaton $\mathcal{A} = \langle L, A, C, \mathcal{E}, inv, l \rangle$, the *region transition system* $\mathcal{R}(\mathcal{A})$ of the automaton $\mathcal{A}$ is a tuple $\langle Q_r, A_r, \mathcal{E}_r, l_r \rangle$, where

- $Q_r = \{\langle \ell, [\nu] \rangle \mid \ell \in L, \nu \in \mathbb{R}^C_{\geqslant 0}\}$,
- $A_r = A \cup \{\lambda\}$,
- $\mathcal{E}_r$ is defined by as follows:
  - $\langle \langle \ell, [\nu] \rangle, \lambda, \langle \ell, succ([\nu]) \rangle \rangle \in \mathcal{E}_r$ iff $succ([\nu]) \models inv(\ell)$,
  - $\langle \langle \ell, [\nu] \rangle, a, \langle \ell', [\nu'] \rangle \rangle \in \mathcal{E}_r$ iff $\langle \ell, \gamma, a, X, \ell' \rangle \in \mathcal{E}$, $\nu \models \gamma$, $\nu' = \nu[X]$ and $\nu' \models inv(\ell')$,
- $l_r(\langle \ell, [\nu] \rangle) = l(\ell) \cup \{\gamma \in \Psi(C) \mid \nu \models \gamma\}$

where $succ([\nu])$ is the *successor equivalence class*, defined as follows [40]: Let α and β be a two equivalence classes such that $\alpha \neq \beta$. We say $succ(\alpha) = \beta$ iff for every $\eta \in \alpha$, there exists $t \in \mathbb{R}_{>0}$ such that for all $t' < t$, $\eta + t \in \beta$ and $\eta + t' \in \alpha \cup \beta$.

Definition 3.6 suggests that the region transition system is constructed by considering timed regions as states. As seen in the definition of $\mathcal{E}_r$, we label all timed transitions with a special symbol λ to represent a time step. We also extend the labelling function l_r with the atomic clock constraints of $\mathcal{A}$ that are true at each location.

3.4. Timed CTL (TCTL)

TCTL [40] is a real-time extension of the branching time logic CTL, where constraints on duration are added to temporal operators. It is a logic, in which timing properties of complex systems can be expressed, something which cannot be done using classical temporal logics. Given that $\sim \in \{<, >, \leqslant, \geqslant, =\}$, $c \in \mathbb{N}$ and $p \in AP$, TCTL is defined according to the following syntax:

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists \varphi_1 \mathcal{U}^{\sim c} \varphi_2 \mid \forall \varphi_1 \mathcal{U}^{\sim c} \varphi_2$$

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 $\exists \Diamond^{\leqslant 5}$ *on* states that "the system will be in the *on* state within 5 time units".

TCTL can be used to express properties of timed automata. Its formulas are interpreted over a timed transition system. Below we define the semantics:

Definition 3.7. Assume $\mathcal{A} = \langle L, A, C, \mathcal{E}, inv, l \rangle$ is a timed automaton, and $\mathcal{T}_\mathcal{A} = \langle S, A_\mathcal{T}, \rightarrow, l_\mathcal{T} \rangle$ is the corresponding timed transition system of $\mathcal{A}$. For a given TCTL-formula φ and a state $s = \langle \ell, \nu \rangle$ in $\mathcal{T}_\mathcal{A}$, the satisfaction relation $\models$ is inductively defined on the structure of φ as follows:

$$\mathcal{T}_\mathcal{A}, s \models p \quad \text{iff} \quad p \in l_\mathcal{T}(s)$$
$$\mathcal{T}_\mathcal{A}, s \models \neg\varphi \quad \text{iff} \quad \mathcal{T}_\mathcal{A}, s \not\models \varphi$$

$$\mathcal{T}_{\mathcal{A}}, s \models \varphi_1 \wedge \varphi_2 \quad \text{iff} \quad \mathcal{T}_{\mathcal{A}}, s \models \varphi_1 \text{ and } \mathcal{T}_{\mathcal{A}}, s \models \varphi_2$$

$$\mathcal{T}_{\mathcal{A}}, s \models \exists \varphi_1 \mathcal{U}^{\sim c} \varphi_2 \quad \text{iff} \quad \exists \sigma \in \mathit{Paths}(s) \text{ with } s = \sigma[0] \text{ and } \exists i \geqslant 0\ s' = \sigma[i]$$
$$\text{s.t. } \mathbb{T}\big(s \overset{\sigma}{\mapsto} s'\big) \sim c, \mathcal{T}_{\mathcal{A}}, s' \models \varphi_2 \text{ and } \big(\forall j < i \text{ with } s'' = \sigma[j]\big)\mathcal{T}_{\mathcal{A}}, s'' \models \varphi_1$$

$$\mathcal{T}_{\mathcal{A}}, s \models \forall \varphi_1 \mathcal{U}^{\sim c} \varphi_2 \quad \text{iff} \quad \forall \sigma \in \mathit{Paths}(s) \text{ with } s = \sigma[0] \text{ and } \exists i \geqslant 0\ s' = \sigma[i]$$
$$\text{s.t. } \mathbb{T}\big(s \overset{\sigma}{\mapsto} s'\big) \sim c, \mathcal{T}_{\mathcal{A}}, s' \models \varphi_2 \text{ and } \big(\forall j < i \text{ with } s'' = \sigma[j]\big)\mathcal{T}_{\mathcal{A}}, s'' \models \varphi_1$$

We say that $\mathcal{A} \models \varphi$ iff $\mathcal{T}_{\mathcal{A}}, \langle \ell, \nu_0 \rangle \models \varphi$, where $\langle \ell, \nu_0 \rangle \in S$ is the initial state [40].

3.5. Probabilistic timed automata

Probabilistic timed automata are an extension of timed automata with discrete probability distributions, or an extension of Markov decision processes with clocks [42]. They are used to model systems that exhibit probabilistic behaviour, such as communication protocols, physical sensors, complex network systems, etc. Probabilistic timed automata are different from timed automata in the sense that transitions between locations can be *both* non-deterministic and probabilistic. In addition to the time aspect existing in timed automata, probabilistic timed automata provide a probability distribution to make a probabilistic choice among the set of enabled edges [43].

Below we give a definition of probabilistic timed automata, slightly different to that of [43] in the sense that we add actions providing a probabilistic distribution over transitions. Therefore, our definition of probabilistic timed automata is more general and more expressive, because we extend timed automata with actions, which is not the case in [43].

Definition 3.8. A *probabilistic timed automaton* (PTA) is a tuple $\langle L, A, C, \mathcal{E}, \mathit{inv}, \mu, l \rangle$, where

- L is a finite set of locations;
- A is a finite set of actions;
- C is a finite set of clocks;
- $\mathcal{E} \subseteq L \times \Psi(C) \times A \times 2^C \times L$ is a set of edges between locations;
- $\mathit{inv} : L \to \Psi(C)$ is a function associating each location with some clock constraints in $\Psi(C)$;
- $\mu : \mathcal{E} \to [0, 1]$ is a *probability transition function*,[2] s.t., for all $\ell \in L$ and $a \in A$ $\sum_{e \in \mathcal{E}_\ell^a} \mu(e) = 1$ where $\mathcal{E}_\ell^a$ denotes the edges from ℓ with the action a is chosen, i.e., $\mathcal{E}_\ell^a = \{\langle \ell, \gamma, a, X, \ell' \rangle \in \mathcal{E}\}$;
- $l : L \to 2^{AP}$ is a labelling function (AP is a set of atomic propositions).

The semantics of probabilistic timed automata is defined in terms of a *probabilistic timed transition system*. All the notations defined in Section 3.3 are also used for PTAs.

Definition 3.9. A *probabilistic timed transition system* $\mathcal{P}_{\mathcal{A}}$ of the probabilistic timed automaton $\mathcal{A} = \langle L, A, C, \mathcal{E}, \mathit{inv}, \mu, l \rangle$ is a continuous time Markov decision process $\langle S, A_{\mathcal{T}}, \to, l_{\mathcal{T}} \rangle$, where

- $S = \{\langle \ell, \nu \rangle \mid \ell \in L,\ \nu \in \mathbb{R}^C_{\geqslant 0} \text{ s.t. } \nu \models \mathit{inv}(\ell)\}$,
- $A_{\mathcal{T}} = \{\langle a, \mu \rangle \mid a \in A\} \cup \mathbb{R}_+$,
- $\to \subseteq S \times A_{\mathcal{T}} \times S$ can be defined in either of the rules below:
  - $\langle \ell, \nu \rangle \overset{d}{\to} \langle \ell, \nu + d \rangle$ iff $\exists d \in \mathbb{R}_+$ s.t. $\nu + d' \models \mathit{inv}(\ell)$, $\forall (0 \leqslant d' \leqslant d)$,
  - $\langle \ell, \nu \rangle \overset{\langle a, \mu \rangle}{\to} \langle \ell', \nu' \rangle$ iff $e = \langle \ell, \gamma, a, X, \ell' \rangle \in \mathcal{E}$, $\mu(e) > 0$, $\nu \models \gamma$, $\nu' = \nu[X]$ and $\nu' \models \mathit{inv}(\ell')$,
- $l_{\mathcal{T}}(\langle \ell, \nu \rangle) = l(\ell) \cup \{\gamma \in \Psi(C) \mid \nu \models \gamma\}$.

Each path of $\mathcal{P}_{\mathcal{A}}$ corresponds to a possible behaviour of $\mathcal{A}$, and it is defined as an infinite sequence $s_0 \overset{\xi_0}{\to} s_1 \overset{\xi_1}{\to} s_2 \overset{\xi_2}{\to} \cdots$, where $\xi_i \in \{\langle a, \mu \rangle \mid a \in A\} \cup \mathbb{R}_+$ ($i \in \mathbb{N}$). Executions of $\mathcal{A}$ can be obtained by resolving both non-deterministic and probabilistic choices, which can be achieved using *schedulers*. This process is not the subject of this article, so we refer the reader to [43] for a detailed discussion.

We now define a finite-state *region transition system* as in Section 3.3. The idea of constructing region graphs for probabilistic timed automata is very similar to that of timed automata. However, in the probabilistic case, the region graphs are in the form of *Markov decision processes* [43]. We transfer the concepts and properties introduced for the construction of region graphs for timed automata.

[2] In order to keep the syntax simple, we consider only one probability distribution μ. It is straightforward to modify our definition to accommodate a finite set of probability distributions $\prod = \{\mu_1, \ldots, \mu_n\}$.

Definition 3.10. Given a probabilistic timed automaton $\mathcal{A} = \langle L, A, C, \mathcal{E}, inv, \mu, l\rangle$, the *region transition system* $\mathcal{R}(\mathcal{A})$ of the automaton $\mathcal{A}$ is a tuple $\langle Q_r, A_r, \mathcal{E}_r, \mu_r, l_r\rangle$, where

- $Q_r = \{\langle \ell, [\nu]\rangle \mid \ell \in L, \nu \in \mathbb{R}^C_{\geqslant 0}\}$,
- $A_r = A \cup \{\lambda\}$,
- $\mathcal{E}_r$ is defined as follows:
  - $\langle\langle \ell, [\nu]\rangle, \lambda, \langle \ell, succ([\nu])\rangle\rangle \in \mathcal{E}_r$ iff $succ([\nu]) \models inv(\ell)$,
  - $\langle\langle \ell, [\nu]\rangle, a, \langle \ell', [\nu']\rangle\rangle \in \mathcal{E}_r$ iff $\langle \ell, \gamma, a, X, \ell'\rangle \in \mathcal{E}$, $\nu \models \gamma$, $\nu' = \nu[X]$ and $\nu' \models inv(\ell')$,
- for every $e_r \in \langle\langle \ell, [\nu]\rangle, \alpha, \langle \ell', [\nu']\rangle\rangle \in \mathcal{E}_r$

$$\mu_r(e_r) = \begin{cases} 1 & \text{if } \alpha = \lambda \\ \frac{\sum_{e \in Adms(\langle \ell, \nu\rangle, a, \langle \ell', \nu'\rangle)} \mu(e)}{\sum_{e \in Adms(\langle \ell, \nu\rangle, a)} \mu(e)} & \text{if } \alpha = a \end{cases}$$

- $l_r(\langle \ell, [\nu]\rangle) = l(\ell) \cup \{\gamma \in \Psi(C) \mid \nu \models \gamma\}$,

where

$$Adms(\langle \ell, \nu\rangle, a) = \{\langle \ell, \gamma, a, X, \bar{\ell}\rangle \in \mathcal{E} \mid \nu \models \gamma \text{ and } \nu[X] \models inv(\bar{\ell})\}$$
$$Adms(\langle \ell, \nu\rangle, a, \langle \ell', \nu'\rangle) = \{\langle \ell, \gamma, a, X, \ell'\rangle \in \mathcal{E} \mid \nu \models \gamma, \nu' = \nu[X] \text{ and } \nu' \models inv(\ell')\}$$ [44].

Informally speaking, $Adms(\langle \ell, \nu\rangle, a)$ is the set of all admissible edges from the location ℓ by executing the action a, and $Adms(\langle \ell, \nu\rangle, a, \langle \ell', \nu'\rangle)$ is the set of all admissible edges from ℓ to ℓ' through executing a. Note that the definitions $Adms(\langle \ell, \nu\rangle, a)$ and $Adms(\langle \ell, \nu\rangle, a, \langle \ell', \nu'\rangle)$ mark different sets. The former considers the edges to all target locations which are reached by resetting some clocks; the latter considers the edges to the target location ℓ' which is reached by executing the action a and resetting some clock set. So, the latter is a subset of the former.

As in Section 3.3, we label all timed transitions with a special symbol λ representing time steps. We also extend the labelling function l_r with the atomic clock constraints of $\mathcal{A}$ which are true at each location.

4. Generic model checking for modal, temporal, and probabilistic logics

In this section, we generalise the model checking algorithm for combined logics given in [25] to cover more complex cases, such as *knowledge*, *belief*, and *probability*. As we have seen in the previous section, the semantics of these logics is defined over more general structures than standard Kripke structures. In our extended model checking algorithm, different model types of combined logics are therefore assumed to have an appropriate hybrid model type.

In the sequel, the following general definition can be used for a propositional logic. (We assume a propositional extension of modal logic.) The language is built from a countable signature of propositional letters $AP = \{p_1, p_2, \ldots\}$, the boolean connectives $\wedge$ and $\neg$, a set of operators $OP = \{O_1^{i_1}, \ldots, O_n^{i_n}\}$ with arities $i_1, \ldots, i_n$ (for $n \in \mathbb{N}$), respectively, and the following formation rules:

- every propositional letter $p \in AP$ is a formula;
- if φ_1, φ_2 are formulas, so are $\neg\varphi_1$ and $\varphi_1 \wedge \varphi_2$;
- if $O_k^{i_k} \in OP$ and $\varphi_1, \ldots, \varphi_{i_k}$ are formulas, so are $O_k^{i_k}(\varphi_1, \ldots, \varphi_{i_k})$ (for $k \in \mathbb{N}$).

The other boolean connectives, such as $\vee$, $\rightarrow$, and $\leftrightarrow$, and the constants $\bot$ and $\top$ can be defined as usual. Depending on the type of the logic, e.g., temporal, probabilistic, belief etc., the semantics is defined over different structures, such as *Kripke structures*, *Markov chains*, *interpreted systems*, etc.

In this section, we assume models with *explicit* sets of states, such as Kripke structures, with a discrete labelling function. In Section 5, we will discuss the case of models with *symbolic states*, such as timed automata. For simplicity, we do not denote initial states in the structure of a model. This can be added without any problem.

Assume a *frame* is a Kripke structure without a valuation function, i.e., a pair $\langle S, T\rangle$ where S is a non-empty set of states, and T is a relation defined on S. Given that $\langle S_1, T_1\rangle$ is a frame for the logic A and $\langle S_2, T_2\rangle$ is a frame for the logic B, a model $\mathcal{M}$ for A(B) is a tuple $\langle S_1 \cup S_2, T_1, T_2, l\rangle$. Note that T_1 (resp. T_2) is an n-tuple denoting a relation with an arbitrary (possibly empty) *labelling* of states and *accessibility* relations on states. Since we consider A and B to be any modal logic, we assume that T can be more general than the accessibility relation, which is usually denoted by R. Namely, T_1 (resp. T_2) gets a different form according to the type of logic A (resp. B). For example, if A (resp. B) is a *modal* logic, then T_1 (resp. T_2) is defined simply as the *accessibility relation* R. If, however, A (resp. B) is a *probabilistic* logic, T_1 (resp. T_2) will be the transition relation with *probability* labels on edges. In Section 4.4 we show how T_1 (resp. T_2) can have different forms according to the type of the logic. We note that within temporalisation, $l : S_2 \rightarrow 2^{AP}$ is a labelling function assigning S_2 to sets of proposition letters.

For a given finite A(B)-model $\mathcal{M}$ and A(B)-formula φ, the model checking problem for A(B) is to check whether there exists $s \in S_1 \cup S_2$ such that $\mathcal{M}, s \models_{\text{A(B)}} \varphi$. Let $\mathbf{MC}_\text{A}$ and $\mathbf{MC}_\text{B}$ be model checkers for the logics A and B, respectively.

$\mathbf{MC}_{\mathrm{AB}}$

Input: $\mathcal{M} = \langle S, T_1, T_2, l\rangle$ and AB formula φ

for every $\psi \in \mathsf{MSSub}(\varphi)$ (increasing order of $|\psi|$)

case(ψ)

$\psi = p \in AP$	:	$Sat(\psi) := \{s \in S \mid p \in l(s)\}$
$\psi = \psi_1 \wedge \psi_2$	:	$Sat(\psi) := \{s \in S \mid p_{\psi_1} \in l(s) \wedge p_{\psi_2} \in l(s)\}$;
$\psi = \neg\psi_1$	:	$Sat(\psi) := \{s \in S \mid p_{\psi_1} \notin l(s)\}$
$\psi = \mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$	:	$Sat(\psi) := \mathbf{MC}_{\mathrm{A}}(\langle S, T_1, l\rangle, \mathbf{O}_{\mathrm{A}}(p_{\psi_1}, \ldots, p_{\psi_n}))$
$\psi = \mathbf{O}_{\mathrm{B}}(\psi_1, \ldots, \psi_n)$	:	$Sat(\psi) := \mathbf{MC}_{\mathrm{B}}(\langle S, T_2, l\rangle, \mathbf{O}_{\mathrm{B}}(p_{\psi_1}, \ldots, p_{\psi_n}))$

for every $s \in Sat(\psi)$, set $l(s) := l(s) \cup \{p_\psi\}$

Fig. 1. Model checking algorithm $\mathbf{MC}_{\mathrm{AB}}$ for the combination of the logics A and B.

The algorithm in Fig. 1 can be used for model checking the combined logic A(B) with the inputs $\mathcal{M}$ and φ. The procedure first computes $\mathsf{MSSub}(\varphi)$, a set of *maximal state* subformulas of φ. The maximal state subformulas is the smallest set of subformulas of φ that satisfies the following constraints:

- φ is in $\mathsf{MSSub}(\varphi)$,
- if ψ' is in $\mathsf{MSSub}(\varphi)$ and ψ' does not start with an operator of OP_{A} then the maximal subformulas of ψ' that do start with an OP_{A} operator is in $\mathsf{MSSub}(\varphi)$,
- if ψ' is in $\mathsf{MSSub}(\varphi)$ and ψ' does not start with an operator of OP_{B} then the maximal subformulas of ψ' that do start with an OP_{B} operator is in $\mathsf{MSSub}(\varphi)$,
- the atomic propositions that occur in φ are in $\mathsf{MSSub}(\varphi)$,
- if ψ is in $\mathsf{MSSub}(\varphi)$ and $\neg\psi$ is a subformula of φ, then $\neg\psi$ is in $\mathsf{MSSub}(\varphi)$, and
- if ψ and ψ' are in $\mathsf{MSSub}(\varphi)$ and $\psi \wedge \psi'$ is a subformula of φ, then $\psi \wedge \psi'$ is in $\mathsf{MSSub}(\varphi)$.

For example, consider the A(B) formula $\forall$ (true $\mathcal{U}(\exists\Diamond\ \exists\Box\ p)$), where $OP_{\mathrm{A}} = \{\forall, \mathcal{U}\}$ and $OP_{\mathrm{B}} = \{\exists, \Diamond, \Box\}$. According to the definition, the maximal state subformulas of A and B are $\{\forall\mathcal{U}\}$ and $\{\exists\Diamond\ \exists\Box, p\}$, respectively. Note that $\exists\Box$ is a not a maximal state subformula. The maximal state subformulas of the logic A are designed to keep the number of calls to the component model checkers low.

The procedure then model checks formulas in $\mathsf{MSSub}(\varphi)$ in increasing order of length of subformulas and extends the labelling l within $\mathcal{M}$ accordingly. The parse tree of φ is recursively descended. For each maximal state subformula ψ of φ, the satisfaction set $Sat(\psi)$, representing the set of states where ψ holds, is calculated. For propositional formulas $p \in AP$, the procedure is trivial. Formulas, whose main operator is in the language of the logics A resp. B are resolved by taking advantage of the corresponding model checker. Prior to calling this model checker, the input to the model checker is modified: for formulas $\mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$ and $\mathbf{O}_{\mathrm{B}}(\psi_1, \ldots, \psi_n)$, every subformula ψ_i (for $1 \leqslant i \leqslant n$) is replaced by a new proposition p_{ψ_i}. This is simply because ψ_i can be either from A or B; therefore we cannot apply $\mathbf{MC}_{\mathrm{A}}$ to $\mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$ if a subformula ψ_i is a B-formula (and vice versa).

Note that the $\mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$ and $\mathbf{O}_{\mathrm{B}}(\psi_1, \ldots, \psi_n)$ are not necessarily operators from the component logics; they are rather formula trees in the component logics. For boolean formulas, e.g., $\psi = \psi_1 \wedge \psi_2$, the satisfaction set of the formula ψ is calculated according to the set of states where p_{ψ_1} and p_{ψ_2} hold.

In Fig. 1, we assume model checkers take a model and a formula ψ as input and return the set of states at which ψ holds.

4.1. Fusion (independent combination)

Let A and B be two propositional logics. A structure $\mathcal{M}$ for $\mathrm{A} \oplus \mathrm{B}$ is a tuple $\langle S_1 \cup S_2, T_1, T_2, l\rangle$, where T_1 (resp. T_2) is a tuple whose elements depend on the type of A (resp. B). Given that $\mathcal{M}$ is a finite $\mathrm{A} \oplus \mathrm{B}$-structure and φ is an $\mathrm{A} \oplus \mathrm{B}$-formula, the model checking problem for $\mathrm{A} \oplus \mathrm{B}$ is to check whether there exists $s \in S_1 \cup S_2$ such that $\mathcal{M}, s \models_{\mathrm{A}\oplus\mathrm{B}} \varphi$. The algorithm in Fig. 1 can again be used for model checking $\mathrm{A} \oplus \mathrm{B}$ with the input $\mathcal{M}$ and φ.

Remark 1. Note that temporalisation is a special case of fusion.

4.2. Join

Let A and B be two propositional logics. Assume $\langle S_1 \times S_2, T, l\rangle$ is a model for the product logic $\mathrm{A} \otimes \mathrm{B}$. Here T is a relation on $S_1 \times S_2$, from which we will derive T_1 and T_2 denoting the corresponding relation for the logic A and B, respectively. T_1 and T_2 have different form depending on the type of the logic. Namely,

(i) if A (resp. B) is a modal or temporal logic without actions, then given that $T = R = (R_1 \cup R_2)$, $T_1 = R_1$ (resp. $T_2 = R_2$), where R_1 (resp. R_2) denotes a binary relation;
(ii) if A (resp. B) is a modal or temporal logic with actions, then given that $T = \langle A = (A_1 \cup A_2), \mathcal{E} = (\mathcal{E}_1 \cup \mathcal{E}_2)\rangle$, $T_1 = \langle A_1, \mathcal{E}_1\rangle$ (resp. $T_2 = \langle A_2, \mathcal{E}_2\rangle$), where A_1 (resp. A_2) is the set of actions for the logic A (resp. B), and $\mathcal{E}_1$ (resp. $\mathcal{E}_2$) is the corresponding set of edges for the logic A (resp. B);
(iii) if A (resp. B) is a probabilistic logic, then given that $T = \langle A = (A_1 \cup A_2), \mathcal{E} = (\mathcal{E}_1 \cup \mathcal{E}_2), \mu\rangle$, $T_1 = \langle A_1, \mathcal{E}_1, \mu_1\rangle$ (resp. $T_2 = \langle A_2, \mathcal{E}_2, \mu_2\rangle$), where μ_1 (resp. μ_2) is a probability transition function.

These cases are formally expressed as follows:

$$
\begin{array}{ll}
\textit{case (i)} & T_1 = R_1, \quad \text{where} \\
& R_1 = \left\{ e \mid e = \left\langle \langle s_1, s\rangle, \langle s_1', s\rangle \right\rangle \in R \text{ s.t. } s_1 \neq s_1' \right\} \\
\textit{case (ii)} & T_1 = \langle A_1, \mathcal{E}_1\rangle, \quad \text{where} \\
& \mathcal{E}_1 = \left\{ e \middle| e = \left\langle \langle s_1, s\rangle, a_1, \langle s_1', s\rangle \right\rangle \in \mathcal{E} \text{ s.t. } s_1 \neq s_1' \text{ and } a_1 \in A_1 \right\} \\
\textit{case (iii)} & T_1 = \langle A_1, \mathcal{E}_1, \mu_1\rangle, \quad \text{where} \\
& \mathcal{E}_1 = \left\{ e \mid e = \left\langle \langle s_1, s\rangle, a_1, \langle s_1', s\rangle \right\rangle \in \mathcal{E} \text{ s.t. } s_1 \neq s_1' \text{ and } a_1 \in A_1 \right\} \\
& \mu_1(e) = \mu(e) \quad \text{for every } e \in \mathcal{E}_1 \text{[3]}
\end{array}
$$

T_2 is calculated in a similar way. Given that $\mathcal{M}$ is a finite A $\otimes$ B-structure and φ is an A $\otimes$ B-formula, the model checking problem for A $\otimes$ B is to check whether there exists a $\langle s_1, s_2\rangle \in S_1 \times S_2$ such that $\mathcal{M}, \langle s_1, s_2\rangle \models_{\mathrm{A}\otimes\mathrm{B}} \varphi$. The algorithm in Fig. 1 can again be used for model checking A $\otimes$ B with inputs $\mathcal{M}$ and φ.

Remark 2. Note that we assume we are given a two-dimensional model $\langle S_1 \times S_2, T, l\rangle$, from which we calculate the input $\mathcal{M} = \langle S_1 \times S_2, T_1, T_2, l\rangle$ to the model checker. Therefore, on top of the cost of model checking on $\mathcal{M}$, there will be an overhead cost for constructing $\mathcal{M}$.

Remark 3. Here we do not consider *handshaking* (*synchronisation*) of the action sets A_1 and A_2. Therefore, we assume $A_1 \cap A_2 = \emptyset$. We consider the case $A_1 \cap A_2 \neq \emptyset$ in Section 5.2.

4.3. Correctness and complexity

Theorem 4.1 *(Termination). Let $\mathcal{M} = \langle S, T_1, T_2, l\rangle$ be a finite structure for the combined logic* AB. *Assume the model checkers* $\mathbf{MC}_{\mathrm{A}}$ *and* $\mathbf{MC}_{\mathrm{B}}$ *are terminating. Then, the combined model checker* $\mathbf{MC}_{\mathrm{AB}}$ *as defined in Fig. 1 also terminates.*

Proof. $\mathbf{MC}_{\mathrm{AB}}$ computes the set of formulas $\mathrm{MSSub}(\varphi)$ in finite time. The procedure also extends the labelling function l that maps each state to a set of proposition letters in finite time. Since the model checkers $\mathbf{MC}_{\mathrm{A}}$ and $\mathbf{MC}_{\mathrm{B}}$ are also terminating, the combined model checker $\mathbf{MC}_{\mathrm{AB}}$ terminates. □

Theorem 4.2 *(Correctness). Let $\mathcal{M} = \langle S, T_1, T_2, l\rangle$ be a finite model for the combination of the logics* A *and* B, *and φ be a combined formula. Assume the model checkers* $\mathbf{MC}_{\mathrm{A}}$ *and* $\mathbf{MC}_{\mathrm{B}}$ *are sound, complete and terminating. Then, $p_\varphi \in l(s)$ if, and only if, $\mathcal{M}, s \models_{\mathrm{AB}} \varphi$, where $s \in S$.*

Proof. We show by induction over the structure of φ that, for every $s \in S$ that $p_\psi \in l(s)$ holds if, and only if, $\mathcal{M}, s \models_{\mathrm{AB}} \psi$. The base cases, $\psi = p$ ($p \in AP$), is obvious.

For the induction step, the cases of boolean combinations, $\psi = \neg\psi_1$ and $\psi = \psi_1 \wedge \psi_2$, of maximal state formulas is trivial. The induction step for the remaining composed modal operators is as follows.

$\psi = \mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$: By induction hypothesis, $p_{\psi_i} \in l(s_i)$ iff $\mathcal{M}, s_i \models_{\mathrm{AB}} \psi_i$ holds for all $1 \leqslant i \leqslant n$ and all $s_i \in S$. Let $\psi' = \mathbf{O}_{\mathrm{A}}(p_{\psi_1}, \ldots, p_{\psi_n})$ be a formula obtained by replacing the ψ_i in $\mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$ by p_{ψ_i}.
The soundness and completeness of the model checker $\mathbf{MC}_{\mathrm{A}}$ provides $p_\psi \equiv p_{\mathbf{O}_{\mathrm{A}}(\psi_1,\ldots,\psi_n)} \in l(s)$ *iff* $\langle S_1, T_1, l\rangle, s \models_{\mathrm{A}} \psi'$.
With $\langle S_1, T_1, l\rangle, s \models_{\mathrm{A}} \psi'$ *iff* $\mathcal{M}, s \models_{\mathrm{AB}} \psi$, we obtain $p_\psi \in l(s)$ *iff* $\mathcal{M}, s \models_{\mathrm{AB}} \psi$.
$\psi = \mathbf{O}_{\mathrm{B}}(\psi_1, \ldots, \psi_n)$: Similar to above case. □

We now analyse the computational complexity of the model checker $\mathbf{MC}_{\mathrm{AB}}$. The complexity of the combined model checker is the sum of *component model checking cost*, which is the cost of performing component model checks, and *interaction processing cost*, which is the sum of the cost of processing inputs and outputs of the component model checks and the cost of operations on the labelling function.

[3] Note that it is trivial to observe that $\sum_{e \in \mathcal{E}_{i_q}^a} \mu_i(e) = 1$ for $i \in \{1, 2\}$ ($\mathcal{E}_{i_q}^a$ denotes the edges from q with the action a is chosen, i.e., $\mathcal{E}_{i_q}^a = \{\langle q, a, q'\rangle \in \mathcal{E}_i\}$).

Theorem 4.3 *(Time complexity). Let $\mathcal{M} = \langle S, T_1, T_2, l\rangle$ be a finite model for the combination of the logics* A *and* B*, φ be a combined formula, and $\mathbb{C}_A$ and $\mathbb{C}_B$ be the time complexities of the model checkers* $\mathbf{MC}_A$ *and* $\mathbf{MC}_B$*, respectively. Then, provided that $\mathbb{C}_A$ and $\mathbb{C}_B$ are at least linear in the size of the specification, we can model check the validity of φ in $\mathcal{M}$ in time*

$$\mathcal{O}\big(\max\{\mathbb{C}_A(|\mathcal{M}|, |\varphi|), \mathbb{C}_B(|\mathcal{M}|, |\varphi|)\}\big)$$

Proof. We first calculate the time required for model checking, which is bounded by $\sum_{\psi\in\mathsf{MSSub}(\varphi)} \mathbb{C}_X(|\mathcal{M}|, |\widehat{\psi}|)$, where X is the appropriate model checker for the respective subformula $\psi \in \mathsf{MSSub}(\varphi)$, and $\widehat{\psi}$ is the simplified subformula that the model checker is called with. (I.e., $\widehat{\psi}$ is the formula obtained from ψ by replacing each true subformulas $\psi' \in \mathsf{MSSub}(\varphi)$ of ψ by $p_{\psi'}$.)

By a simple inductive argument along the structure of the formula, we can show that the sum of the length $\sum_{\psi\in\mathsf{MSSub}(\varphi)} |\widehat{\psi}| < 2|\varphi|$.

For $m = \max\{\mathbb{C}_A(|\mathcal{M}|, |\varphi|), \mathbb{C}_B(|\mathcal{M}|, |\varphi|)\}$, this provides us—together with the trivial assumption that the cost of model checking is at least linear in the size of the specification—with the straight forward estimation $\sum_{\psi\in\mathsf{MSSub}(\varphi)} \mathbb{C}_X(|\mathcal{M}|, |\widehat{\psi}|) < 2m$.

We now calculate the interaction processing cost. The computation time of $\mathsf{MSSub}(\varphi)$ is linear in the size of φ, and is therefore in $\mathcal{O}(|\varphi|)$. The cost factor of computing or updating the labelling functions l depends on the form of ψ. According to the model checking algorithm, l is updated $\sum_{\psi\in\mathsf{MSSub}(\varphi)} \sum_{s\in S} 1$ times. The outer sum is iterated $\mathcal{O}(|\varphi|)$ times, and the inner sum $|S|$ times. Hence, the processing time of l is in $\mathcal{O}(|\varphi|.|S|)$. This implies that the interaction processing cost is dominated by the component model checking cost.

The total cost of the combined model checking is therefore in $\mathcal{O}(\max\{\mathbb{C}_A(|\mathcal{M}|, |\varphi|), \mathbb{C}_B(|\mathcal{M}|, |\varphi|)\})$. □

Theorem 4.4 *(Concise representation). If storing the model and the results concisely provide a complexity advantage, then this advantage is inherited.*

Proof. The correctness of the algorithm does not depend on the representation. □

Examples for using conciseness are more often referring to space complexity. Region graphs are simple examples, where the explicit representation as a region graph is exponential in the abstract representation as a timed automaton. However, if the fusion (or product) refers to locations of the timed automaton (not distinguishing different states of the region graph that refer to the same location), then a model checking algorithm can maintain and extend the labelling function on a more abstract timed automaton. With proofs similar to those of the time complexity, we can establish the same claim for space complexity.

Theorem 4.5 *(Space complexity). Let $\mathcal{M} = \langle S, T_1, T_2, l\rangle$ be a finite model for the combination of the logics* A *and* B*, φ be a combined formula, and $\mathbb{C}_A$ and $\mathbb{C}_B$ be the space complexities of the model checkers* $\mathbf{MC}_A$ *and* $\mathbf{MC}_B$*, respectively. Then, provided that $\mathbb{C}_A$ and $\mathbb{C}_B$ are at least linear in the size of the specification, we can model check the validity of φ in $\mathcal{M}$ in space*

$$\mathcal{O}\big(\max\{\mathbb{C}_A(|\mathcal{M}|, |\varphi|), \mathbb{C}_B(|\mathcal{M}|, |\varphi|)\}\big)$$

Remark 4. Although $\mathbf{MC}_A$ and $\mathbf{MC}_B$ are not applied to the whole model $\mathcal{M}$, we give the complexity of the generic model checking in terms of $|\mathcal{M}|$. In fact, as we can see in Fig. 1, $\mathbf{MC}_A$ and $\mathbf{MC}_B$ are applied to $\langle S_1, T_1, l\rangle$ and $\langle S_2, T_2, l\rangle$, respectively, which are smaller then $\mathcal{M}$. We calculate the complexity in terms of $|\mathcal{M}|$ in order to keep modularity and simplicity.

Remark 5. We note that the combined model checking cost of 'join' is in general higher than that of either 'temporalisation' or 'fusion'. First of all, the state space of the combined model is the product of state spaces of the component logics while in 'temporalisation' and 'fusion' the combined state space is the union of two constituent state spaces. Secondly, the model checking of join has an extra *overhead* cost while calculating T_1 and T_2 (see Section 4.2). This cost does not change the overall complexity, but requires extra computation time.

4.4. Example combinations

In this section, we show how the method can be applied to represent some combinations of different aspects of multi-agent systems. The examples demonstrate how simple it becomes with our technique to determine the model checking complexity of these logics (and to synthesise a model checker for them).

4.4.1. Time + knowledge

Let us consider the combination of time and knowledge aspects. A typical example is the *fusion* of the temporal logic CTL with the modal logic $S5_n$. Typically, the $S5_n$ modal operator is used to formalise epistemic concepts. Thus, the combination $\mathrm{CTL} \oplus S5_n$ can be used to reason about knowledge and time.

Assume the modalities 'group knowledge', 'common knowledge' and 'distributed knowledge' are already defined in $S5_n$. The language of CTL $\oplus$ $S5_n$ is the smallest set of formulas generated by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists \bigcirc \varphi \mid \exists\Box\varphi \mid \exists[\varphi_1 \mathcal{U} \varphi_2] \mid K_i\varphi \mid E_\Gamma\varphi \mid C_\Gamma\varphi \mid D_\Gamma\varphi$$

Other operators ($\exists\Diamond$, $\forall\bigcirc$, $\forall\Diamond$, $\forall\Box$, $\forall\mathcal{U}$, etc.) can be derived in the usual way. In this grammar, $p \in AP$ is an atomic proposition, and the operators $\exists\bigcirc$, $\exists\Box$ and $\exists\mathcal{U}$ are the standard CTL operators. Given a set of agents $\mathbb{A} = \{1, \ldots, n\}$, for any agent $i \in \mathbb{A}$, the formula $K_i\varphi$ is read as "agent i knows φ". Given that $\Gamma \subseteq \mathbb{A}$ denotes a group of agents, the form formula $E_\Gamma\varphi$ is read as "everybody in group Γ knows φ"; the formula $C_\Gamma\varphi$ is read as "φ is common knowledge in group Γ"; and the formula $D_\Gamma\varphi$ is read as "φ is distributed knowledge in group Γ".

Assume the combined model $\mathcal{M}$ for the logic CTL $\oplus$ $S5_n$ is a tuple $\langle S, R_t, \sim_1, \ldots, \sim_n, l\rangle$, where $S = S_1 \times \cdots \times S_n$ is a set of global states, $R_t \subseteq S \times S$ is a temporal relation, $\sim_i \subseteq S \times S$ is an epistemic accessibility relation for i, and $l : S \to 2^{AP}$. Here we assume that $\langle s, s'\rangle \in R_t$ iff s' is the result of applying a transition function $t : S \times A \to S$ (where $A \subseteq A_1 \times \cdots \times A_n$ is the set of joint actions) to the global state s and a joint action $a \in A$. As can easily be observed, $\mathcal{M}$ is an interpreted system, $\langle S, R_t\rangle$ is a Kripke frame for CTL, and $\langle S, \sim_1, \ldots, \sim_n\rangle$ is a Kripke frame for $S5_n$.

Given that σ is a *path*, which is an infinite sequence of states, and $\sigma[i]$ is the ith element of σ, the semantics of CTL $\oplus$ $S5_n$ is defined inductively by:

$$\mathcal{M}, s \models p \quad \textit{iff } p \in l(s), \text{ for } p \in AP$$

$$\mathcal{M}, s \models \neg\varphi \quad \textit{iff not } \mathcal{M}, s \models \varphi$$

$$\mathcal{M}, s \models \varphi_1 \wedge \varphi_2 \quad \textit{iff } \mathcal{M}, s \models \varphi_1 \text{ and } \mathcal{M}, s \models \varphi_2$$

$$\mathcal{M}, s \models \exists \bigcirc \varphi \quad \textit{iff for some } \sigma \in \mathit{Paths}(s),\ \sigma[0] = s \text{ and } \mathcal{M}, \sigma[1] \models \varphi$$

$$\mathcal{M}, s \models \exists\Box\varphi \quad \textit{iff for some } \sigma \in \mathit{Paths}(s),\ \sigma[0] = s \text{ and } (\forall i \geqslant 0)\mathcal{M}, \sigma[i] \models \varphi$$

$$\mathcal{M}, s \models \exists[\varphi_1 \mathcal{U} \varphi_2] \quad \textit{iff for some } \sigma \in \mathit{Paths}(s),\ \sigma[0] = s \text{ and } (\exists k \geqslant 0)$$
$$\text{s.t. } \mathcal{M}, \sigma[k] \models \varphi_2 \text{ and } \forall(0 \leqslant j < k)\mathcal{M}, \sigma[j] \models \varphi_1$$

$$\mathcal{M}, s \models K_i\varphi \quad \textit{iff } (\forall s' \in S)\ s \sim_i s' \text{ implies } \mathcal{M}, s' \models \varphi$$

$$\mathcal{M}, s \models E_\Gamma\varphi \quad \textit{iff } (\forall s' \in S)\ s \sim_\Gamma^E s' \text{ implies } \mathcal{M}, s' \models \varphi$$

$$\mathcal{M}, s \models C_\Gamma\varphi \quad \textit{iff } (\forall s' \in S)\ s \sim_\Gamma^C s' \text{ implies } \mathcal{M}, s' \models \varphi$$

$$\mathcal{M}, s \models D_\Gamma\varphi \quad \textit{iff } (\forall s' \in S)\ s \sim_\Gamma^D s' \text{ implies } \mathcal{M}, s' \models \varphi$$

where $\sim_\Gamma^E = \bigcup_{i\in\Gamma} \sim_i$, $\sim_\Gamma^D = \bigcap_{i\in\Gamma} \sim_i$, and $\sim_\Gamma^C = (\sim_\Gamma^E)^+$ (the transitive closure of $\sim_\Gamma^E$).

According to our combined model checking procedure, the model checking complexity of CTL $\oplus$ $S5_n$ transfers the complexity of the component logics CTL and $S5_n$. It is trivial to observe that the model checking problem of CTL $\oplus$ $S5_n$ can be solved in polynomial time.

This result is interesting, because without introducing a complex method we could solve the model checking problem of a combined logic of time and knowledge. Basically, our result provides an upper bound for the model checking problem of the epistemic logic of branching time, CTLK [26]. As can be easily seen, CTL $\oplus$ $S5_n$ and CTLK are the same logics, because they have the same grammar and same semantics over interpreted systems. The model checking problem of the logic CTLK was previously analysed by bounded model checking [26], by unbounded model checking [4], and by building upon an extension of the model checker NuSMV [14], and the same result was found.

Above we have applied our method to combine knowledge and branching time aspects. We can also use our approach to combine knowledge and linear time aspects. Namely, we can define the fusion of the logic LTL and $S5_n$, which will result in the same language of the logic CKL_n [7]. We can easily show that the complexity of the model checking LTL $\oplus$ $S5_n$ is the same as that of LTL, because the complexity of the LTL model checking is higher than the complexity of the $S5_n$ model checking. This is the same result as that of [7], where the model checking problem of CKL_n is reduced to LTL model checking by a method that uses *local propositions*. The advantage of our approach is that we can prove the model checking complexity of a combined logic just using the model checking complexities of component logics.

4.4.2. Probability + knowledge

In the case of merging the probability and knowledge dimensions of multi-agent systems, we can combine probabilistic CTL, PCTL [39], with the modal epistemic logic S5. PCTL $\oplus$ S5 allows us to express probabilistic, temporal and epistemic properties.

Let $\mathbb{A} = \{1, \ldots, n\}$ be a set of agents. The syntax of PCTL $\oplus$ S5 is given according to the following syntax:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Pr_{\sim r}[\psi] \mid K_i\varphi \mid E_\Gamma\varphi \mid C_\Gamma\varphi \mid D_\Gamma\varphi$$

$$\psi ::= \varphi \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U}^{\leqslant k} \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2$$

where $\sim \in \{<, >, \leqslant, \geqslant, =\}$, $r \in [0,1]$, $k \in \mathbb{N}$, $i \in \mathbb{A}$, $\Gamma \subseteq \mathbb{A}$, $p \in AP$ is an atomic proposition, $\Pr_{\sim r}$ is the probability operator, K_i, E_Γ, C_Γ and D_Γ are epistemic operators as described in Section 4.4.1, and $\mathcal{U}, \mathcal{U}^{\leqslant k}$ are path formulas.

The combined model for the logic PCTL $\oplus$ S5 is a tuple $\langle S, A, \mathcal{E}, \mu, \sim_1, \ldots, \sim_n, l\rangle$, where $S = S_1 \times \cdots \times S_n$ is a set of global states, A is a set of actions, $\mathcal{E}$ is a set of edges and μ is the probability transition function (see Section 3.2), $\sim_i \subseteq S \times S$ is an epistemic accessibility relation for i, and $l : S \to 2^{AP}$. The semantics of the probabilistic formula $\Pr_{\sim r}[\psi]$ and the path formulas $\bigcirc\varphi$, $\varphi_1 \mathcal{U}^{\leqslant k} \varphi_2$, $\varphi_1 \mathcal{U} \varphi_2$ are defined as in Section 3.2; and the semantics of the epistemic formulas $K_i\varphi, E_\Gamma\varphi, C_\Gamma\varphi, D_\Gamma\varphi$ are defined as in Section 4.4.1.

According to our combined model checking procedure, PCTL $\oplus$ S5 transfers complexities from the component logics PCTL and S5. We know that PCTL model checking over a Markov chain has polynomial complexity [45]. Therefore, we can find a polynomial upper bound for the model checking of the combined logic PCTL $\oplus$ S5. This is interesting because PCTL $\oplus$ S5 subsumes the logic KPCTL, which was shown to have polynomial model checking complexity [23]. Thus, with our method we could establish the same complexity result without devising a new model checking algorithm.

Remark 6. In this section, we have shown how our method can be used to combine different dimensions, such as time, knowledge and probability. Actually, this approach can be applied to typical extensions of modal logic. For example, in Section 6 we apply the combination method to Hybrid Logics [46], an extension of modal logic with a reference to locations, called *nominals*.

5. Generic model checking for modal, temporal, probabilistic, and real-time logics

In the cases "time + knowledge" and "probability + knowledge", we considered models with *explicit* states, and therefore we could consider model checkers as mapping states to sets of subformulas of φ. However, model checkers for real-time logics, in general, receive models with *non-explicit* states. For example, in timed automata such states are called *locations*, which are just abstract representations. The explicit states can be viewed as the tuples of locations and clock valuations. Therefore, we can no longer consider such a model checker labelling locations with subformulas that are true.

In this section, we generalise the model checking algorithm from Section 4 to cover real-time logics. As mentioned above, the semantics of all these different logics are defined over different structures. In the extended model checking algorithm, we should therefore either combine different model types of combined logics into a single structure and deal with hybrid model types, or define a generic model for all different types of logics. The former approach requires extending algorithms in such a way that they deal with the combination of different types of models. However, this makes algorithms unnecessarily complex and impractical. We therefore chose the latter approach and use a generic model type for different types of models. For the case of combining modal, temporal, probabilistic, and real-time logics, *probabilistic timed automata* (*PTA*) can be used as a generic structure, because PTA embed Kripke structures, Markov decision processes and timed automata.

5.1. Fusion

In this section, we consider a model checking procedure for the combination (fusion) of A and B, which are any arbitrary modal, temporal, probabilistic, or real-time logics. In order to employ real-time logics whose models include non-explicit states, we must modify the algorithm from Fig. 1.

Since we need a generic structure that covers all the types of logics mentioned, we assume that the fusion A $\oplus$ B is interpreted over a probabilistic timed automaton $\mathcal{A} = \langle L, A, C, \mathcal{E}, inv, \mu, l\rangle$, where $\mathcal{A}$ is the *union* of the probabilistic timed automata $\mathcal{A}_1 = \langle L_1, A_1, C_1, \mathcal{E}_1, inv_1, \mu_1, l_1\rangle$ and $\mathcal{A}_2 = \langle L_2, A_2, C_2, \mathcal{E}_2, inv_2, \mu_2, l_2\rangle$. Given that $A_1 \cap A_2 = \emptyset$ and $C_1 \cap C_2 = \emptyset$, we define $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ as follows.

- $L = L_1 \cup L_2$,
- $A = A_1 \cup A_2$,
- $C = C_1 \cup C_2$,
- $inv(\ell) = inv_1(\ell) \cup inv_2(\ell)$ for every $\ell \in L$,
- $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$,
- $\mu(e) = \begin{cases} \mu_1(e) & \text{if } e \in \mathcal{E}_1, \\ \mu_2(e) & \text{if } e \in \mathcal{E}_2, \end{cases}$
- $l(\ell) = l_1(\ell) \cup l_2(\ell)$ for every $\ell \in L$.

$\mathcal{E}$ includes the edges $\langle \ell, \gamma_1, a_1, X_1, \ell'\rangle \in \mathcal{E}_1$ and $\langle \ell, \gamma_2, a_2, X_2, \ell'\rangle \in \mathcal{E}_2$. Since the clock sets X_1 and X_2 are disjoint, the clock constraints γ_1 and γ_2 cannot refer to the same clocks.

Let φ be a A $\oplus$ B-formula and $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ be a A $\oplus$ B-model. The algorithm in Fig. 2, with the inputs $\mathcal{A}$, A_1, A_2 and φ, can be used for model checking $\mathcal{A} \models_{\text{A}\oplus\text{B}} \varphi$.

In Fig. 2, we first construct the region graph $\mathcal{R}(\mathcal{A} \oplus z, \varphi) = \langle Q_r, A_r, \mathcal{E}_r, \mu_r, l_r\rangle$ as defined in Definition 3.10. Every state $q \in Q_r$ is a tuple $\langle \ell, [\nu]\rangle$, where $\ell \in L$ is a location and $[\nu]$ is an equivalence class. $\mathcal{E}_r$ consists of two types of transitions: (i) transitions representing the passage of time; (ii) transitions representing the edges in probabilistic timed automata. Here,

$\mathbf{MC}_{\mathrm{AB}}$

Input: $\mathcal{A} = \langle L, A, C, \mathcal{E}, inv, \mu, l\rangle$, A_1, A_2 and AB formula φ

compute $\mathcal{R}(\mathcal{A} \oplus z, \varphi) := \langle Q_r, A_r, \mathcal{E}_r, \mu_r, l_r\rangle$

compute $\mathcal{R}_1 := \langle Q_r, A_{r_1}, \mathcal{E}_{r_1}, \mu_{r_1}, l_r\rangle$
compute $\mathcal{R}_2 := \langle Q_r, A_{r_2}, \mathcal{E}_{r_2}, \mu_{r_2}, l_r\rangle$

for every $\psi \in \mathrm{MSSub}(\varphi)$ (increasing order of $|\psi|$)

case(ψ)
$\psi = p \in AP$: $Sat(\psi) := \{q \in Q_r \mid p \in l_r(q)\}$
$\psi = \psi_1 \wedge \psi_2$: $Sat(\psi) := \{q \in Q_r \mid p_{\psi_1} \in l_r(q) \wedge p_{\psi_2} \in l_r(q)\}$;
$\psi = \neg\psi_1$: $Sat(\psi) := \{q \in Q_r \mid p_{\psi_1} \notin l_r(q)\}$
$\psi = \mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$: $Sat(\psi) := \mathbf{MC}_{\widehat{\mathrm{A}}}(\mathcal{R}_1, \mathbf{O}_{\widehat{\mathrm{A}}}(p_{\psi_1}, \ldots, p_{\psi_n}))$
$\psi = \mathbf{O}_{\mathrm{B}}(\psi_1, \ldots, \psi_n)$: $Sat(\psi) := \mathbf{MC}_{\widehat{\mathrm{B}}}(\mathcal{R}_2, \mathbf{O}_{\widehat{\mathrm{B}}}(p_{\psi_1}, \ldots, p_{\psi_n}))$
for every $q \in Sat(\psi)$, set $l_r(q) := l_r(q) \cup \{p_\psi\}$

Fig. 2. Generic model checking algorithm for the combination of modal, temporal, probabilistic and real-time logics.

μ_r is the probability distribution of the region graph, which is calculated as described in Definition 3.10. It is easy to observe that the region graph $\mathcal{R}$ is actually a Markov decision process.

Note that automaton $\mathcal{A}$ is extended with a new clock variable $z \notin C$, that the clock constraints within the formula φ refers to. $\mathcal{R}(\mathcal{A} \oplus z, \varphi)$ actually depends on the formula φ. This is simply because the maximal constants of φ are taken into the consideration when calculating equivalence classes. If $\varphi = true$ or the maximal constants in φ do not exceed the maximal constants of the clock constraints of the automaton, then $\mathcal{R}(\mathcal{A} \oplus z, \varphi)$ reduces to $\mathcal{R}(\mathcal{A} \oplus z)$. Otherwise, the labelling function l_r should also include the clock constraints of φ. We recall that while constructing region transition systems in Sections 3.3 and 3.5, we considered atomic clock constraints of (probabilistic) timed automata as atomic propositions, and updated the labelling function accordingly. Since $\mathcal{R}(\mathcal{A} \oplus z, \varphi)$ depends on φ, the definition of l_r (in Definition 3.10) is modified as follows (here Ψ_φ denotes the set of clock constraints of φ):

$$l_r\big(\langle \ell, [\nu]\rangle\big) = l(\ell) \cup \big\{\gamma \in \Psi(C) \;\big|\; \nu \models \gamma\big\} \cup \{\gamma \in \Psi_\varphi \mid \nu \models \gamma\}$$

The model checking algorithm then calculates the graphs $\mathcal{R}_1 := \langle Q_r, A_{r_1}, \mathcal{E}_{r_1}, \mu_{r_1}, l_r\rangle$ and $\mathcal{R}_2 := \langle Q_r, A_{r_2}, \mathcal{E}_{r_2}, \mu_{r_2}, l_r\rangle$, where

- $A_{r_1} = \mathcal{A}_r \setminus (A_2 \setminus A_1)$,
- $A_{r_2} = \mathcal{A}_r \setminus (A_1 \setminus A_2)$,
- $\mathcal{E}_{r_1} = \mathcal{E}_r \setminus \{\langle\langle \ell, [\nu]\rangle, a, \langle \ell', [\nu']\rangle\rangle \in \mathcal{E}_r \mid a \in A_2 \setminus A_1\}$,
- $\mathcal{E}_{r_2} = \mathcal{E}_r \setminus \{\langle\langle \ell, [\nu]\rangle, a, \langle \ell', [\nu']\rangle\rangle \in \mathcal{E}_r \mid a \in A_1 \setminus A_2\}$,
- $\mu_{r_1}(e_{r_1}) = \mu_r(e_{r_1})$ for every $e_{r_1} \in \mathcal{E}_{r_1}$,
- $\mu_{r_2}(e_{r_2}) = \mu_r(e_{r_2})$ for every $e_{r_2} \in \mathcal{E}_{r_2}$.

Informally speaking, $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) is a model for the first (resp. second) component logic where the edges labelled with the actions of the second (resp. first) logic are abstracted from.

Note that any subformula ψ of the form of $\mathbf{O}_{\mathrm{A}}(\psi_1, \ldots, \psi_n)$ (resp. $\mathbf{O}_{\mathrm{B}}(\psi_1, \ldots, \psi_n)$) is translated into the formula $\widehat{\psi} = \mathbf{O}_{\widehat{\mathrm{A}}}(\psi_1, \ldots, \psi_n)$ (resp. $\mathbf{O}_{\widehat{\mathrm{B}}}(\psi_1, \ldots, \psi_n)$) by eliminating timing constraints in ψ. Then, the model checking $\mathbf{MC}_{\widehat{\mathrm{A}}}$ (resp. $\mathbf{MC}_{\widehat{\mathrm{B}}}$) is called for $\mathbf{O}_{\widehat{\mathrm{A}}}(p_{\psi_1}, \ldots, p_{\psi_n})$ (resp. $\mathbf{O}_{\widehat{\mathrm{B}}}(p_{\psi_1}, \ldots, p_{\psi_n})$). Here, $\widehat{\mathrm{A}}$ (resp. $\widehat{\mathrm{B}}$) refers to the corresponding logic of A (resp. B), where timing parameters are removed. If there is no timing parameter, we assume $\mathrm{A} \equiv \widehat{\mathrm{A}}$ (resp. $\mathrm{B} \equiv \widehat{\mathrm{B}}$) and $\psi \equiv \widehat{\psi}$.

We eliminate the timing parameters by simply extending the probabilistic timed automaton $\mathcal{A}$ under investigation with a new clock variable z, which is not included in the clock set of $\mathcal{A}$. The extended automaton is denoted by $\mathcal{A} \oplus z$ (i.e., z is added to the clock set of $\mathcal{A}$). The new clock z is used to measure the elapse of time until a subformula of the given formula ψ holds. The basic idea is that the clock constraints of ψ are extended with z so that we obtain the corresponding atomic propositions referring to z. Namely, we remove any clock constraint, e.g., $\sim c$, in an operator, e.g., $\mathcal{U}^{\sim c}$, and obtain the atomic proposition $z \sim c$ and the *untimed* operator $\mathcal{U}$.

We exemplify this idea on the example of TCTL. Assume $\psi = \mathbf{P}(\psi_1 \mathcal{U}^{\sim c} \psi_2)$ is a TCTL formula, where $\mathbf{P} \in \{\exists, \forall\}$ is a path quantifier. If we eliminate the timing parameters, then we obtain the formula $\widehat{\psi} = \mathbf{P}((\psi_1 \vee \psi_2)\mathcal{U}(z \sim c) \wedge \psi_2)$, which is the corresponding CTL formula. Namely, the clock constraint $\sim c$ is eliminated from the TCTL formula ψ, and the formula $z \sim c$, which is the extension of the clock constraint with z, is added as an atomic proposition in the corresponding timing parameter-free CTL formula $\widehat{\psi}$. For example, the TCTL formula $\exists\Diamond^{<1}\varphi$ is replaced by the CTL formula $\exists\Diamond((z < 1) \wedge \varphi)$.

5.2. Product (join)

The fusion of A and B results in a restricted logic. Although it is computationally not (much) harder, the *product* of two logics results in a much richer language. Assume that $\mathcal{A}_1 = \langle L_1, A_1, C_1, \mathcal{E}_1, inv_1, \mu_1, l_1\rangle$ and $\mathcal{A}_2 = \langle L_2, A_2, C_2, \mathcal{E}_2, inv_2, \mu_2, l_2\rangle$

are probabilistic timed automata with $C_1 \cap C_2 = \emptyset$.[4] Given that $\mathfrak{A} \subseteq A_1 \cap A_2$ is a set of *handshaking* actions, the probabilistic timed automaton $\mathcal{A} = \mathcal{A}_1 \| _{\mathfrak{A}} \mathcal{A}_2$, which is the *parallel composition* automaton of $\mathcal{A}_1$ and $\mathcal{A}_2$, is defined as $\langle L, A, C, \mathcal{E}, inv, \mu, l \rangle$ where

- $L = L_1 \times L_2$,
- $A = A_1 \cup A_2$,
- $C = C_1 \cup C_2$,
- $inv(\langle \ell_1, \ell_2 \rangle) = inv_1(\ell_1) \wedge inv_2(\ell_2)$ for every $\langle \ell_1, \ell_2 \rangle \in L$,
- $\mathcal{E} = \mathcal{E}^0 \cup \mathcal{E}^1 \cup \mathcal{E}^2$ s.t.
  - for $a \in \mathfrak{A}$, $e = \langle \langle \ell_1, \ell_2 \rangle, \gamma_1 \wedge \gamma_2, a, X_1 \cup X_2, \langle \ell'_1, \ell'_2 \rangle \rangle \in \mathcal{E}^0$ *iff* $e_1 = \langle \ell_1, \gamma_1, a, X_1, \ell'_1 \rangle \in \mathcal{E}_1$ and $e_2 = \langle \ell_2, \gamma_2, a, X_2, \ell'_2 \rangle \in \mathcal{E}_2$ (denoted by $e = e_1 \| _a e_2$),
  - for $a \notin \mathfrak{A}$, $e = \langle \langle \ell_1, \ell_2 \rangle, \gamma_1, a, X_1, \langle \ell'_1, \ell_2 \rangle \rangle \in \mathcal{E}^1$ *iff* $e_1 = \langle \ell_1, \gamma_1, a, X_1, \ell'_1 \rangle \in \mathcal{E}_1$ and $\nexists \ell'_2 \in L_2$ s.t. $e_2 = \langle \ell_2, \gamma_2, a, X_2, \ell'_2 \rangle \in \mathcal{E}_2$ (denoted by $e = e_1 \not\| _a e_2$),
  - for $a \notin \mathfrak{A}$, $e = \langle \langle \ell_1, \ell_2 \rangle, \gamma_2, a, X_2, \langle \ell_1, \ell'_2 \rangle \rangle \in \mathcal{E}^2$ *iff* $e_2 = \langle \ell_2, \gamma_2, a, X_2, \ell'_2 \rangle \in \mathcal{E}_2$ and $\nexists \ell'_1 \in L_1$ s.t. $e_1 = \langle \ell_1, \gamma_1, a, X_1, \ell'_1 \rangle \in \mathcal{E}_1$ (denoted by $e = e_2 \not\| _a e_1$),
- $\mu(e) = \begin{cases} \mu_1(e_1) \times \mu_2(e_2) & \text{if } e = e_1 \| _a e_2, \\ \mu_1(e_1) & \text{if } e = e_1 \not\| _a e_2, \\ \mu_2(e_2) & \text{if } e = e_2 \not\| _a e_1, \end{cases}$
- $l(\langle \ell_1, \ell_2 \rangle) = l_1(\ell_1) \cup l_2(\ell_2)$ for every $\langle \ell_1, \ell_2 \rangle \in L$.

The combined model checking procedure $\mathbf{MC}_{\mathrm{AB}}$ in Fig. 2 can be used to model check an A ⊗ B formula φ over a parallel composite automaton $\mathcal{A} = \mathcal{A}_1 \| _{\mathfrak{A}} \mathcal{A}_2$. $\mathbf{MC}_{\mathrm{AB}}$ takes $\mathcal{A}$, A_1, A_2 and φ as input, and returns whether $\mathcal{A}$ satisfies φ.

5.3. Correctness and complexity

Let $\mathcal{M} = \langle L, A, C, \mathcal{E}, inv, \mu, l \rangle$ be a composition automaton (either union or product), $\mathcal{P}_{\mathcal{M}}$ be the corresponding probabilistic timed transition system (as defined in Definition 3.9) and $\mathcal{R} = \langle Q_r, A_r, \mathcal{E}_r, \mu_r, l_r \rangle$ be the corresponding region graph (as defined in Definition 3.10).

Theorem 5.1 *(Termination). Let $\mathcal{M}$ be a model for the combination of the logics* A *and* B*. Assume the model checkers $\mathbf{MC}_{\widehat{\mathrm{A}}}$ and $\mathbf{MC}_{\widehat{\mathrm{B}}}$ are terminating. Then, the combined model checker $\mathbf{MC}_{\mathrm{AB}}$ also terminates.*

Proof. $\mathbf{MC}_{\mathrm{AB}}$ computes the region graph $\mathcal{R}$, and the graphs $\mathcal{R}_1, \mathcal{R}_2$ in finite time. The calculation of subformulas $\mathsf{MSSub}(\varphi)$ and the update of the labelling function l are also bounded, because the number of iterations is bounded by finite sets. Since the model checkers $\mathbf{MC}_{\widehat{\mathrm{A}}}$ and $\mathbf{MC}_{\widehat{\mathrm{B}}}$ are also terminating, the combined model checker $\mathbf{MC}_{\mathrm{AB}}$ terminates. □

We now prove the correctness of the model checking algorithm. We first define the following lemma.

Lemma 5.2. *Let ψ be a formula, and $\mathcal{A}$ be a probabilistic timed automaton. The state $\langle \ell, \nu \rangle$ of the corresponding probabilistic timed transition system $\mathcal{P}_{\mathcal{A}}$ satisfies ψ if, and only if, the vertex $\langle \ell, [\nu] \rangle$ of the corresponding region graph $\mathcal{R}(\mathcal{A}, \psi)$ satisfies the formula $\widehat{\psi}$, where $\widehat{\psi}$ is derived by eliminating timing constraints in ψ.*

Proof. If ψ is a formula of a logic that does not contain any timing constraint, then no clock valuation and equivalence class is defined, and $\mathcal{A}$, $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{R}(\mathcal{A}, \psi)$ become similar. Also, ψ and $\widehat{\psi}$ become the same formula. It is then trivial to observe that the lemma is true.

If ψ is a formula of a logic with timing constraints, then clock valuations and equivalence classes are defined. The lemma is proven in [40] for the case of TCTL, and in [43] for the case of PTCTL. □

In this article, we consider TCTL and PTCTL as logics of timing constraints. We therefore consider these two logics in the lemma above. The proof methods presented in [40] and [43] apply to similar logics.

Theorem 5.3 *(Correctness). Let $\mathcal{M}$ be a model for* AB *and φ be an* AB *formula. Assume the model checkers $\mathbf{MC}_{\widehat{\mathrm{A}}}$ and $\mathbf{MC}_{\widehat{\mathrm{B}}}$ are sound, complete and terminating. Then $p_\varphi \in l_r(\langle \ell, [\nu] \rangle)$ if, and only if, $\mathcal{P}_{\mathcal{M}}, \langle \ell, \nu \rangle \models_{\mathrm{AB}} \varphi$, where $\langle \ell, [\nu] \rangle \in Q_r$.*

Proof. We show by induction over the structure of φ that, for every $\psi \in \mathsf{MSSub}(\varphi)$ and $\langle \ell, [\nu] \rangle \in Q_r$, $p_\psi \in l_r(\langle \ell, [\nu] \rangle)$ holds if, and only if, $\mathcal{P}_{\mathcal{M}}, \langle \ell, \nu \rangle \models_{\mathrm{AB}} \psi$. The base cases, $\psi = p$ ($p \in AP$), is obvious.

[4] Here, we assume that the clock sets of the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint, because if there is a transition between these two automata (this is the case, for example, at intersection nodes), this clock will be forced to be reset. This will result in an unexpected situation as a common clock, which should not be affected from transitions between automata, will be reset at each transition.

For the induction step, the cases of boolean combinations, $\psi = \neg\psi_1$ and $\psi = \psi_1 \wedge \psi_2$, of maximal state formulas is trivial. The induction step for the remaining composed modal operators is as follows.

$\psi = \mathbf{O}_{\mathsf{A}}(\psi_1, \ldots, \psi_n)$: By induction hypothesis we have that $p_{\psi_i} \in l_r(\langle \ell_i, [\nu_i]\rangle)$ *iff* $\mathcal{P}_{\mathcal{M}}, \langle \ell_i, [\nu_i]\rangle \models_{\mathsf{AB}} \psi_i$, for $1 \leqslant i \leqslant n$.
Let $\mathcal{R}_1 = \langle Q_r, A_{r_1}, \mathcal{E}_{r_1}, \mu_{r_1}, l_r\rangle$, and let $\mathcal{P}_1$ be the corresponding probabilistic timed transition system of $\mathcal{R}_1$. Since $\mathbf{MC}_{\widehat{\mathsf{A}}}$ is sound and complete, by the combined model checking procedure we have that $p_\psi \equiv p_{\mathbf{O}_{\mathsf{A}}(\psi_1,\ldots,\psi_n)} \in l_r(\langle \ell, [\nu]\rangle)$ *iff* $\mathcal{R}_1, \langle \ell, [\nu]\rangle \models_{\widehat{\mathsf{A}}} \mathbf{O}_{\widehat{\mathsf{A}}}(p_{\psi_1}, \ldots, p_{\psi_n})$.
By Lemma 5.2, $\mathcal{R}_1, \langle \ell, [\nu]\rangle \models_{\widehat{\mathsf{A}}} \mathbf{O}_{\widehat{\mathsf{A}}}(p_{\psi_1}, \ldots, p_{\psi_n})$ *iff* $\mathcal{P}_1, \langle \ell, \nu\rangle \models_{\mathsf{A}} \mathbf{O}_{\mathsf{A}}(p_{\psi_1}, \ldots, p_{\psi_n})$. Together, this implies that $p_\psi \equiv p_{\mathbf{O}_{\mathsf{A}}(\psi_1,\ldots,\psi_n)} \in l_r(\langle \ell, [\nu]\rangle)$ *iff* $\mathcal{P}_1, \langle \ell, \nu\rangle \models_{\mathsf{A}} \mathbf{O}_{\mathsf{A}}(p_{\psi_1}, \ldots, p_{\psi_n})$.
We know that the graph $\mathcal{R}_1$ is calculated by eliminating the transitions with the actions from $A_2 \setminus A_1$, i.e., $\mathcal{E}_{r_1} = \mathcal{E}_r \setminus \{\langle\langle \ell, [\nu]\rangle, a, \langle \ell', [\nu']\rangle\rangle \in \mathcal{E}_r \mid a \in A_2 \setminus A_1\}$ (see Section 5.1). So, we have that $\mathcal{E}_{r_1} \subseteq \mathcal{E}_r$. Note that the probability distributions are defined in Definition 3.9 such that the sum of probabilities of edges per action is 1, and eliminating the transitions with the actions from $A_2 \setminus A_1$ in $\mathcal{R}_1$ does not affect the probability distributions of $\mathcal{R}$ the remaining edges. Thus, the property of being a probability distribution is not affected for the remaining actions and the $\mathcal{R}_1$ is defined properly. Furthermore, in the definition of $\mathcal{R}_1$ we have for any edge $e_1 \in \mathcal{E}_1$ that the corresponding transition $e_{r_1} \in \mathcal{E}_{r_1}$ of the graph $\mathcal{R}_1$ provides $\mu_{r_1}(e_{r_1}) = \mu_r(e_{r_1})$.
From this observation we can conclude that $\mathcal{P}_1, \langle \ell, \nu\rangle \models_{\mathsf{A}} \mathbf{O}_{\mathsf{A}}(p_{\psi_1}, \ldots, p_{\psi_n})$ *iff* $\mathcal{P}_{\mathcal{M}}, \langle \ell, \nu\rangle \models_{\mathsf{AB}} \mathbf{O}_{\mathsf{A}}(p_{\psi_1}, \ldots, p_{\psi_n})$. Hence, $p_\psi \in l_r(\langle \ell, [\nu]\rangle)$ *iff* $\mathcal{P}_{\mathcal{M}}, \langle \ell, \nu\rangle \models_{\mathsf{AB}} \psi$.
$\psi = \mathbf{O}_{\mathsf{B}}(\psi_1, \ldots, \psi_n)$: Similar to the above case. □

We now analyse the computational complexity of the model checker $\mathbf{MC}_{\mathsf{AB}}$.

Theorem 5.4 *(Complexity). Let $\mathcal{M}$ be a model for AB, φ be a* AB *formula, and $\mathbb{C}_{\widehat{\mathsf{A}}}$ and $\mathbb{C}_{\widehat{\mathsf{B}}}$ be the time (or space) complexities of the model checkers $\mathbf{MC}_{\widehat{\mathsf{A}}}$ and $\mathbf{MC}_{\widehat{\mathsf{B}}}$, respectively, where $\widehat{\mathsf{A}}$ (resp.* $\widehat{\mathsf{B}}$*) refers to the corresponding logic of* A *(resp.* B*) where timing parameters are removed. Then, provided that $\mathbb{C}_{\widehat{\mathsf{A}}}$ and $\mathbb{C}_{\widehat{\mathsf{B}}}$ are at least linear in the size of the specification, we can model check the validity of φ in $\mathcal{M}$ in*

$$\mathcal{O}\big(\max\{\mathbb{C}_{\widehat{\mathsf{A}}}(|\mathcal{R}|, |\varphi|), \mathbb{C}_{\widehat{\mathsf{B}}}(|\mathcal{R}|, |\varphi|)\}\big)$$

time (or space), where (given that for $x \in C$, c_x be the largest constant that is compared in some clock constraint in $\mathcal{M}$ or in a formula φ), where

$$|\mathcal{R}| \in \mathcal{O}\left(|\mathcal{M}|.|C|!.\prod_{x\in C} c_x\right)$$

Proof. We first calculate the cost of constructing the region graphs. Assume L is the set of locations, $\mathcal{E}$ is the set of edges and C is the set of clocks of a timed automaton $\mathcal{A}$. [40] proves that the size of the corresponding region graph R is in $\mathcal{O}((|L| + |\mathcal{E}|).|C|!.\prod_{x\in C} c_x)$. Probabilistic timed automata also include probability distribution μ; but the addition of probability distributions to times automata does not significantly increase the size of region graph over the size of non-probabilistic region graph [43]. So, for a probabilistic timed automaton $\mathcal{A}$, we have that $|R| \in \mathcal{O}(|\mathcal{A}|.|C|!.\prod_{x\in C} c_x)$.

From this result we can derive that the size of $\mathcal{R}$ is in $\mathcal{O}(|\mathcal{M}|.|C|!.\prod_{x\in C} c_x)$.[5] Since $|\mathcal{R}_1| < |\mathcal{R}|$ and $|\mathcal{R}_2| < |\mathcal{R}|$, the sizes of $\mathcal{R}_1$ and $\mathcal{R}_2$ are also in $\mathcal{O}(|\mathcal{M}|.|C|!.\prod_{x\in C} c_x)$.

$\mathsf{MSSub}(\varphi)$ calculates the maximal state subformulas ψ of φ. By a simple inductive argument along the structure of the formula, we can show that the sum $\sum_{\psi\in\mathsf{MSSub}(\varphi)} |\widehat{\psi}| < 2|\varphi|$ of the length of the simplified subformula $\widehat{\psi}$ the model checker is called with, is bounded by $2m$. ($\widehat{\psi}$ is again the formula obtained from ψ by replacing each true subformulas $\psi' \in \mathsf{MSSub}(\varphi)$ of ψ by $p_{\psi'}$.)

We now consider the cost of updating the labelling function l, which is updated $|\mathsf{MSSub}(\varphi)|.|Q_r|$ times. $|\mathsf{MSSub}(\varphi)|$ is in $\mathcal{O}(|\varphi|)$. For the case of timed automata, [40] proves that the size of Q_r is in $\mathcal{O}(|L|.|C|!.\prod_{x\in C} c_x)$. For probabilistic timed automata, we have that $|Q_r|$ is in $\mathcal{O}(|\mathcal{A}|.|C|!.\prod_{x\in C} c_x)$. Hence, the cost of updating l is in $\mathcal{O}(|\varphi|.|\mathcal{M}|.|C|!.\prod_{x\in C} c_x)$.

We now calculate the time required for component model checking. The total cost of applying $\mathbf{MC}_{\widehat{\mathsf{A}}}$ is $\sum_{\psi\in\mathsf{MSSub}(\varphi)} \mathbb{C}_{\widehat{\mathsf{A}}}(|\mathcal{R}_1|, |\widehat{\psi}|)$. From the assumption that the cost of model checking is at least linear in the size of the specification and the fact $\sum_{\psi\in\mathsf{MSSub}(\varphi)} |\widehat{\psi}| < 2|\varphi|$, we can derive that the cost of $\mathbf{MC}_{\widehat{\mathsf{A}}}$ is in $\mathcal{O}(\mathbb{C}_{\widehat{\mathsf{A}}}(|\mathcal{R}_1|, |\varphi|)) \subset \mathcal{O}(\mathbb{C}_{\widehat{\mathsf{A}}}(|\mathcal{R}|, |\varphi|))$, where $|\mathcal{R}| = \mathcal{O}(|\mathcal{M}|.|C|!.\prod_{x\in C} c_x)$. Similarly, the cost of $\mathbf{MC}_{\widehat{\mathsf{B}}}$ is in $\mathcal{O}(\mathbb{C}_{\widehat{\mathsf{B}}}(|\mathcal{R}|, |\varphi|))$.

We assume that the cost of model checking is at least linear in the size of the specification, which implies that the total cost of constructing region graphs and updating labelling function is dominated by the component model checking costs. Thus, the total cost of the combined model checking is in $\mathcal{O}(\max\{\mathbb{C}_{\widehat{\mathsf{A}}}(|\mathcal{R}|, |\varphi|), \mathbb{C}_{\widehat{\mathsf{B}}}(|\mathcal{R}|, |\varphi|)\})$, where $|\mathcal{R}| = \mathcal{O}(|\mathcal{M}|.|C|!.\prod_{x\in C} c_x)$.

[5] The computation of the successor class *succ* of any equivalence class (see Definition 3.6) requires additional $\mathcal{O}(|C|)$ time. Since this is dominated by the cost of constructing the region graph, we can ignore this cost.

The same argument applies for space. □

As in the modal case, this extends to concise representations.

Theorem 5.5 *(Complexity for concise systems). Let $\mathcal{M}$ be a model for* AB*, φ be a* AB *formula, and $\mathbb{C}_{\widehat{\mathrm{A}}}$ and $\mathbb{C}_{\widehat{\mathrm{B}}}$ be the time (or space) complexities of the model checkers* $\mathbf{MC}_{\widehat{\mathrm{A}}}$ *and* $\mathbf{MC}_{\widehat{\mathrm{B}}}$*, respectively, where $\widehat{\mathrm{A}}$ (resp. $\widehat{\mathrm{B}}$) refers to the corresponding logic of* A *(resp.* B*) where timing parameters are removed. Then, provided that $\mathbb{C}_{\widehat{\mathrm{A}}}$ and $\mathbb{C}_{\widehat{\mathrm{B}}}$ are at least linear in the size of the specification, we can model check the validity of φ in $\mathcal{M}$ in*

$$\mathcal{O}\big(\max\{\mathbb{C}_{\widehat{\mathrm{A}}}(|\mathcal{M}|,|\varphi|),\mathbb{C}_{\widehat{\mathrm{B}}}(|\mathcal{M}|,|\varphi|)\}\big)$$

time (or space), provided that the extended labelling function produced by the algorithm from Fig. 2 only refers to the locations of $\mathcal{M}$.

Proof. In this case, the $Sat(\psi)$ for the formulas in $\mathsf{MSSub}(\varphi)$ is independent of the clock valuations, and it hence suffices to compute them for the case that all clock values are 0. The extended labelling function then refers to the location of the (probabilistic) timed automaton $\mathcal{M}$. The remainder is as in the explicit case. □

Remark 7. Usually, the concise representation cannot be used for product.

Remark 8. Although the complexity bounds for model checking fusion and product of logics are similar, model checking of product requires more computation time, because the state space of the product model is the product of the state spaces of the component logics while in fusion the combined state space is the union of two constituent state spaces.

Remark 9. Note that the model checking algorithm presented in Fig. 2 is the real-time extension of the algorithm in Fig. 1. Therefore, if the component logics are not real-time logics, both algorithms are equivalent.

5.4. Example combinations

Below we analyse different combinations of real-time logics. Here we select TCTL to express real-time clock constraints. Other real-time logics can be analysed along the same lines.

5.4.1. Real-time + modal (temporal) logics

In this section, we consider the combination (fusion) TCTL $\oplus$ X of TCTL with any modal logic X. As described above, TCTL is interpreted over timed automata. A timed automaton can be represented in terms of a probabilistic timed automaton $\mathcal{A}_1=\langle L_1,A_1,C_1,\mathcal{E}_1,inv_1,\mu_1,l_1\rangle$, if we define a probability transition function μ_1 that assigns 1 to the probability of each transition. That is, $\mathcal{A}_1$ with $\mu_1:\mathcal{E}_1\to 1$ represents a timed automaton whose every edge is labelled with the probability of 1. In this way, we can eliminate the probability element from $\mathcal{A}_1$.

The second component of the fusion is the modal logic X, which is assumed to be interpreted over Kripke structures. In order to represent a Kripke structure in terms of a probabilistic timed automaton $\mathcal{A}_2=\langle L_2,A_2,C_2,\mathcal{E}_2,inv_2,\mu_2,l_2\rangle$, similar to the TCTL case, we simply assign 1 to each transition probability, i.e., $\mu_2:\mathcal{E}_2\to 1$. We can eliminate the real-time dimension by defining an empty clock set and an empty clock constraint set, i.e., $C_2=\emptyset$ and $\Psi(C_2)=\emptyset$. However, this is not sufficient to remove the time element from the $\mathcal{A}_2$, because the automaton can wait in any location infinitely. We therefore need to assign $false$ to the invariance of each location, i.e., $inv_2(\ell)=\bot$ for any $\ell\in L_2$. In this way, as soon as the automaton enters a location ℓ, it exists immediately because $inv_2(\ell)$ is never valid. By eliminating the probability and real-time elements, $\mathcal{A}_2$ becomes a Kripke structure, which can be used as a model for the logic X.

Given that $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfy the constraints above, we assume that the union $\mathcal{A}=\mathcal{A}_1\cup\mathcal{A}_2$ is a model for the combined logic. Then, the model checking procedure presented in Fig. 2 can be instantiated to cover the fusion of TCTL $\oplus$ X.

In order to show the usefulness of combining TCTL with any other logic X, we assume X is the branching logic CTL*. The syntax of TCTL $\oplus$ CTL* is defined by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1\wedge\varphi_2 \mid \exists\varphi_1\mathcal{U}^{\sim c}\varphi_2 \mid \forall\varphi_1\mathcal{U}^{\sim c}\varphi_2 \mid \exists\psi \mid \forall\psi$$

$$\psi ::= \varphi \mid \neg\psi \mid \psi_1\wedge\psi_2 \mid \psi_1\mathcal{U}\psi_2$$

where $\sim\in\{<,>,\leqslant,\geqslant,=\}$ and $c\in\mathbb{N}$. Standard abbreviations can be done as usual.

Assume TCTL $\oplus$ CTL* is interpreted over the combined structure $\mathcal{A}=\langle L,A,C,\mathcal{E},inv,l\rangle$, which is the union of $\mathcal{A}_1=\langle L_1,A_1,C_1,\mathcal{E}_1,inv_1,\mu_1,l_1\rangle$ and $\mathcal{A}_2=\langle L_2,A_2,C_2,\mathcal{E}_2,inv_2,\mu_2,l_2\rangle$ with the restrictions stated above; i.e., $\mathcal{A}_1$ is a timed automaton, and $\mathcal{A}_2$ is a Kripke structure. We also assume that $\mathcal{E}_2=\{\langle\ell,\bot,\epsilon,\emptyset,\ell'\rangle\mid\langle\ell,\gamma,a,X,\ell'\rangle\in\mathcal{E}_1\}$. Intuitively, this means that $\mathcal{E}_2$ simulates $\mathcal{E}_1$ by adding untimed edges labelled with dummy action $\epsilon\notin A_1$. We further assume $L_1=L_2=L$. Therefore, in the region graph, for each transition $\langle q,a,q'\rangle\in\mathcal{E}_r$ labelled with an action $a\in A_1$, there will be a transition $\langle q,\epsilon,q'\rangle\in\mathcal{E}_r$ labelled with a dummy action ϵ. This allows the generic model checking algorithm to call the component model checkers on the same state space, and bisimilar graphs.

The logic TCTL$\oplus$CTL* combines the expressive powers of TCTL and CTL*. (The semantics of the modalities can be defined as in Section 3.) It is therefore more expressive than all of the logics LTL, CTL, CTL* and TCTL. TCTL$\oplus$CTL* can express some properties that cannot be expressed in any of these languages. For example, consider the fairness property

> "if there are infinitely many processes p along any path, then the process q cannot continuously run for more than k seconds",

which can be expressed by the TCTL$\oplus$CTL* formula

$$\forall\big[\Box\Diamond p \rightarrow \big(\forall\Box\big(q \rightarrow \forall\Diamond^{>k}\neg q\big)\big)\big]$$

Note that this property cannot be expressed in either LTL, CTL, CTL* or TCTL. Similarly, the TCTL$\oplus$CTL* formula $\forall\Diamond\Box p \vee \forall\Box\exists\Diamond^{\leqslant k} q$ cannot be expressed in any of these logics.

Given that φ is a TCTL$\oplus$CTL* formula, by Theorem 5.4, the algorithmic model checking complexity of TCTL$\oplus$CTL* is calculated as $\mathcal{O}(\max\{\mathbb{C}_{\text{CTL}}(|\mathcal{R}_{\mathcal{A}}|,|\varphi|),\mathbb{C}_{\text{CTL}^*}(|\mathcal{R}_{\mathcal{A}}|,|\varphi|)\})$. Since the model checking complexity of CTL is linear in the size of input, the model checking complexity of the first component is EXPTIME (due to the size of $\mathcal{R}$), which dominates the cost of model checking CTL*, which is PSPACE.

From the model checking complexities of the constituent logics, we might assume that the model checking complexity of TCTL$\oplus$CTL* is EXPTIME. However, we can achieve a better result. The work in [40] proves that the model checking complexity of TCTL is PSPACE. Since we take the fusion of TCTL and CTL*, at the intersection nodes of the combined model (where the timed automaton is entered), the clock values are reset to 0. This means that we can consider these nodes as the initial locations of the timed automaton. Therefore, the correctness of a subformula at an intersection node can be easily assessed by calling the TCTL model checker without a need to create a region graph for the combined model. This suggests that we can only consider the model size of the concise representation, which is the timed automaton itself, when determining the complexity. By Theorem 5.5, since storing the model and the results concisely provide a complexity advantage, we can inherit this complexity advantage. Therefore, we can consider that the TCTL model checking can be done using space *polynomial* in the length of the input (rather than EXPTIME). From this, we can easily derive that the model checking complexity of TCTL$\oplus$CTL* is PSPACE.

This result is important in the sense that extending TCTL with CTL* semantics (thus with LTL semantics), or extending CTL* with timing constraints does not increase the model checking complexities of these logics.

5.4.2. Real-time + knowledge

In this section, we discuss the combination (fusion) of real-time and knowledge. We consider the fusion of the real-time logic TCTL and the logic of knowledge S5. Assume $\mathbb{A}=\{1,\ldots,n\}$ be set of agents. The syntax of the resulting logic TCTL$\oplus$S5 is defined by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid K_i\varphi \mid E_\Gamma\varphi \mid C_\Gamma\varphi \mid D_\Gamma\varphi \mid \exists\varphi_1\mathcal{U}^{\sim c}\varphi_2 \mid \forall\varphi_1\mathcal{U}^{\sim c}\varphi_2$$

where $\sim\in\{<,>,\leqslant,\geqslant,=\}$, $c\in\mathbb{N}$, $i\in\mathbb{A}$, $\Gamma\subseteq\mathbb{A}$, and $p\in AP$ is an atomic proposition. Standard abbreviations include $\bot,\varphi_1\vee\varphi_2,\varphi_1\Rightarrow\varphi_2$, etc. as well as $\exists\Diamond^{\sim c}\varphi$, $\forall\Diamond^{\sim c}\varphi$, $\exists\Box^{\sim c}\varphi$ and $\forall\Box^{\sim c}\varphi$.

Assume TCTL$\oplus$S5 is interpreted over a combined structure $\mathcal{A}=\langle L,A,C,\mathcal{E},inv,\mu,l\rangle$, which is the fusion of $\mathcal{A}_1=\langle L,A,C,\mathcal{E}_1,inv,\bar{\mu},l\rangle$ and $\mathcal{A}_2=\langle L,\emptyset,\emptyset,\emptyset,\bot,\bar{\mu},l\rangle$, where $L=L_1\times\cdots\times L_n$ is a set of global states, $\bar{\mu}$ denotes a probability transition function that assigns 1 to every transition, $\bot$ denotes that the invariance of each location is $false$. It is easy to see that $\mathcal{A}_1$ has no probability dimension and $\mathcal{A}_2$ has no real-time dimension.

In order to employ epistemic relations, we modify the graph $\mathcal{R}_2$ in Fig. 2 as follows:

$$\mathcal{R}_2=\langle Q_r,\sim_1,\ldots,\sim_n,l_r\rangle$$

where Q_r and l_r are defined as before, $\sim_i\subseteq Q_r\times Q_r$ is an epistemic relation defined by $\langle\ell,\nu\rangle\sim_i\langle\ell',\nu'\rangle$ *if, and only of,* $prj_i(\ell)=prj_i(\ell')$ and $\nu\simeq\nu'$, for each agent i and for valuations $\nu,\nu'\in\mathbb{R}^C_{\geqslant 0}$. Here, $prj_i:L\rightarrow L_i$ returns the location of the agent i in the global location. Note that $\mathcal{R}_2$ is an *interpreted system* on Q_r, over which the semantics of epistemic operators are defined.

The semantics of the real-time formulas $\exists\varphi\mathcal{U}^{\sim c}\psi$ and $\forall\varphi\mathcal{U}^{\sim c}\psi$ can be defined as in Section 3.4, and the semantics of the epistemic formulas $K_i\varphi,E_\Gamma\varphi,C_\Gamma\varphi$ and $D_\Gamma\varphi$ are defined as in Section 4.4.1. The model checking procedure presented in Fig. 2 (with the modified definition of $\mathcal{R}_2$) can be used to for model checking TCTL$\oplus$S5 formulas.

The model checking problem of the combination of real-time and epistemic aspects of multi-agent systems was studied in [20], where the fusion of an existential fragment of TCTL and the logic S5, called TECTLK, was introduced. TCTL$\oplus$S5 is a more expressive logic than TECTLK. In [20], it was shown that the model checking problem is decidable; but no complexity result was provided.

Using our combination method, we derive that the algorithmic model checking complexity of TCTL$\oplus$S5 is in EXPTIME. However, as described in Section 5.4.1, we can inherit the complexity advantage for TCTL, which reduces the complexity to PSPACE. This result fills an important gap in the spectrum.

5.4.3. Real-time + probability

In this section, we analyse the combination of real-time and probability dimensions. Namely, we consider the product TCTL ⊗ PCTL[6] of the logics TCTL and PCTL. The syntax of the product is defined inductively as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists\varphi_1 \mathcal{U}^{\sim c}\varphi_2 \mid \forall\varphi_1 \mathcal{U}^{\sim c}\varphi_2 \mid \mathrm{P}_{\sim r}[\varphi_1 \mathcal{U} \varphi_2]$$

where $\sim \in \{<, >, \leqslant, \geqslant, =\}$, $c \in \mathbb{N}$ and $r = [0, 1]$. Standard abbreviations can be done as usual.

Assume TCTL ⊗ PCTL is interpreted over the combined structure $\mathcal{A} = \langle L, A, C, \mathcal{E}, inv, \mu, l\rangle$, which is the product of $\mathcal{A}_1 = \langle L, A, C, \mathcal{E}_1, inv, \bar{\mu}, l\rangle$ and $\mathcal{A}_2 = \langle L, A, \emptyset, \mathcal{E}_2, \bot, \mu, l\rangle$ (where $\bar{\mu}$ denotes a probability transition function that assigns 1 to every transition and $\bot$ denotes that the invariance of each location is *false*). Clearly, $\mathcal{A}_1$ has no probability dimension and $\mathcal{A}_2$ has no real-time dimension. We also assume that $\mathcal{E}_2 = \{\langle \ell, \bot, a, \emptyset, \ell'\rangle \mid \langle \ell, \gamma, a, X, \ell'\rangle \in \mathcal{E}_1\}$. Intuitively, this means that $\mathcal{E}_2$ includes the untimed simulations of the edges of $\mathcal{E}_1$.

The model checking procedure presented in Fig. 2 can be instantiated to cover the product of TCTL ⊗ PCTL. The semantics of the formulas $\exists\varphi_1\mathcal{U}^{\sim c}\varphi_2, \forall\varphi_1\mathcal{U}^{\sim c}\varphi_2, \mathrm{P}_{\sim r}[\varphi_1\mathcal{U}\varphi_2]$ can be found in Section 5.4.1. TCTL ⊗ PCTL allows us to express properties such as "with probability greater than 0.9 the message is delivered within 3 seconds", which can be expressed as $\mathrm{P}_{>0.9}[true\ \mathcal{U}^{\leqslant 3}\ delivered]$. This suggests that we can formalise properties that involve both real-time constraints and probabilistic expressions.

A similar logic was defined in [43], where *Probabilistic Timed* CTL (PTCTL) is introduced. PTCTL includes both the probabilistic and timed operators. Namely, PTCTL has the following syntax:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathrm{P}_{\trianglerighteq r}[\exists\varphi_1\mathcal{U}^{\sim c}\varphi_2] \mid \mathrm{P}_{\trianglerighteq r}[\forall\varphi_1\mathcal{U}^{\sim c}\varphi_2]$$

where $\trianglerighteq \in \{>, \geqslant\}$, $\sim \in \{<, >, \leqslant, \geqslant, =\}$, $c \in \mathbb{N}$, and $r = [0, 1]$. PTCTL formulas are interpreted over probabilistic timed automata.

As can be seen TCTL⊗PCTL has a richer syntax than PTCTL. For example, some TCTL⊗PCTL formulas, such as $\mathrm{P}_{\leqslant r}[\exists\Diamond^{\sim c}\varphi]$ and $\exists\Diamond^{\sim c}(\mathrm{P}_{<r}[true\ \mathcal{U}\varphi])$, cannot be expressed in PTCTL. This makes the combined logic more expressive than its similar counterpart.

Theorem 5.4 implies that the algorithmic model checking complexity of TCTL ⊗ PCTL is EXPTIME. Note that since we consider the product space of the constituent logics, we can no longer take the advantage of the coincide representation unlike the case of fusion. We therefore cannot place the complexity below EXPTIME.

Our complexity analysis shows that TCTL ⊗ PCTL and PTCTL have the same model checking complexities. This shows the usefulness of our combination method, which allows us to calculate the model checking complexity without introducing a new method.

5.4.4. Real-time + probability + knowledge

In this section, we analyse a more complex situation, where we consider the combination of 'real-time', 'probability' and 'knowledge' dimensions. If we extend TCTL ⊗ PCTL with S5, we obtain the combination (TCTL ⊗ PCTL) ⊕ S5. The syntax of the new logic is:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists\varphi_1\mathcal{U}^{\sim c}\varphi_2 \mid \forall\varphi_1\mathcal{U}^{\sim c}\varphi_2 \mid \mathrm{P}_{\sim r}[\varphi_1\mathcal{U}\varphi_2] \mid K_i\varphi \mid E_\Gamma\varphi \mid C_\Gamma\varphi \mid D_\Gamma\varphi$$

where $\sim \in \{<, >, \leqslant, \geqslant, =\}$, $c \in \mathbb{N}$ and $r = [0, 1]$.

Assume TCTL ⊗ PCTL is interpreted over the automaton $\mathcal{A}_1 = \langle L, A, C, \mathcal{E}, inv, \mu, l\rangle$, as in Section 5.4.3. The semantics of (TCTL ⊗ PCTL) ⊕ S5 formulas can be defined on the fusion of the structures $\mathcal{A}_1$ and $\mathcal{A}_2 = \langle L, \emptyset, \emptyset, \emptyset, \bot, \bar{\mu}, l\rangle$, where $L = L_1 \times \cdots \times L_n$. As discussed in Section 5.4.2, we construct an interpreted system $\langle Q_r, \sim_1, \ldots, \sim_n, l_r\rangle$ for the epistemic component of the combination. Here, $\sim_i \subseteq Q_r \times Q_r$ is an epistemic relation defined by $\langle \ell, \nu\rangle \sim_i \langle \ell', \nu'\rangle$ *if, and only if* $prj_i(\ell) = prj_i(\ell')$ and $\nu \simeq \nu'$, for each agent i and for valuations $\nu, \nu' \in \mathbb{R}^C_{\geqslant 0}$.

Theorem 5.4 implies that the model checking complexity of (TCTL ⊗ PCTL) ⊕ S5 is EXPTIME. This result shows that 'knowledge' dimension can be added to the combination of 'real-time' and 'probability' without increasing the complexity.

5.5. Discussion

Above, we have shown how model checkers for the constituent *temporal, probabilistic and real-time logics* can be re-used in a modular way when we consider combined logics involving different dimensions. This avoids the re-implementation of model checking procedures. We have shown that our combination method is very useful in determining model checking complexities of some existing logics by studying their combined counterparts.

Our approach is novel in the sense that:

(i) *we can re-produce some complexity results in the literature:* For example, we have found that the complexity of CTL ⊕ S5 is polynomial, which is same as the complexity of the equivalent logic CTLK [26] (see Table 1). Similarly, the complexity result we have found for the logic LTL ⊕ S5 is PSPACE, which is the same as that of its counterpart CKL_n [7].

[6] We do not include the bounded until operator within the syntax of PCTL in order to prevent confusion with the bound operator in TCTL.

Table 1
Summary of combining different elements and the corresponding complexity results.

Combination	Related logic	Complexity	Remark
CTL $\oplus$ S5	$=$ CTLK	PTIME	confirming [26]
PCTL $\oplus$ S5	$\supseteq$ KPCTL	PTIME	confirming [23] – richer logic
LTL $\oplus$ S5	$=$ CKL_n	PSPACE	confirming [7]
TCTL $\oplus$ CTL*	–	PSPACE	new logic – new complexity result
TCTL $\oplus$ S5	$\supseteq$ TECTLK	PSPACE	richer logic – new complexity result
TCTL $\oplus$ PCTL	$\supseteq$ PTCTL	EXPTIME	confirming [43] – richer logic
(TCTL $\otimes$ PCTL) $\oplus$ S5	–	EXPTIME	new logic – new complexity result

(ii) *we can obtain richer logics without increasing the complexity:* For example, TCTL $\oplus$ S5 is more expressive than TECTLK [20]. Although the complexity problem for TECTLK has not been studied, we have found that the complexity of TCTL $\oplus$ S5 is PSPACE. Another example is the logic PCTL $\oplus$ S5. Although we have found the same complexity as that of KPCTL [23], the former is more expressive than the latter.

(iii) *we can obtain new logics*: For example, by combining different dimensions we have obtained the logics TCTL $\oplus$ CTL* and (TCTL $\otimes$ PCTL) $\oplus$ S5 and studied their complexities. This shows the usability of the combination method to introduce new logics that have not been studied before.

6. Application of the method to a pervasive system

In this section, we apply the combination method to the *Scatterbox* message-forwarding system to demonstrate the usefulness of our generic model checking approach, which allows us to handle the combination of distinct and non-trivial logics.

We will start with a simple model of the Scatterbox system, which can be analysed using separate model checking techniques. We will then refine this simple model into more realistic and more complex versions in order to show that standard techniques cannot easily be used for verification.

6.1. Scatterbox: A message delivery system

The *Scatterbox* system [47] is a message delivery system, which is "a test bed for context-aware computing in a pervasive computing environment". The following is taken from [47]: Scatterbox system provides a content-filtering service to its users by forwarding relevant messages to their mobile phones. The user's context is derived by tracking his/her location and monitoring his/her daily schedule. This context data is analysed, and situations are identified that indicate the user's level of interruptibility. As messages arrive, Scatterbox forwards them to subscribed users provided the user's available context suggests they are in a situation where they may be interrupted. Scatterbox consists of the following components: *Construct*, a distributed, fully decentralised platform supporting the building of context-aware, adaptive, pervasive and autonomous systems; *Bluetooth* and *calendar sensors*, which provide Scatterbox with contextual data; an *e-mail handler* that interacts with an e-mail server to access a user's e-mail and determine each e-mail's importance; a *situation reasoner* that takes these context data and determines whether or not the user is interruptible, and whether a particular message is relevant enough to be forwarded to the user; and the *message delivery component*, which sends relevant messages to the user via either Bluetooth or SMS.

6.2. Modelling the system

The Scatterbox has been formally analysed in [48]. Here we take a different approach.

As an initial step, we model the behaviour of the message forwarding component of the system. Basically, if Scatterbox decides to forward a message, downloaded from the user's e-mail server, to the user's mobile device, the message forwarding component forwards the message either through Bluetooth or SMS. If the user's mobile device is in range of a 'Bluetooth-enabled node', the message is forwarded to the user's device through that node. When it arrives on the user's handset, the user has the opportunity to accept or reject the message. If the device is outside the range of a Bluetooth-enabled node, the message is forwarded through SMS. In this case, the user does not have the option to reject the message.

A simplified model of the message forwarding component is given in Fig. 3, illustrating the probabilistic distribution of Bluetooth sensor usage, and rates of message acceptance, message rejection, and message loss, based on our assumption. In order to model this probabilistic behaviour, we adapt the model, which allows the specification of probabilistic state changes. *Markov chains*, in particular *Discrete Time Markov chains* (*DTMCs*), are a simple and adequate formalism for representing probabilistic behaviour over time.

According to the simple model in Fig. 3, when a message is forwarded to the user's mobile device, 70% of the time this is done via SMS, and 30% of the time is done via Bluetooth. For simplicity we assume that there are three Bluetooth sensors, and the frequency of their usage is the same. When a message is forwarded to the user via Bluetooth, the probabilities of the user accepting and rejecting the message is 0.5 and 0.4, respectively; the probability of the message being lost is 0.1. On

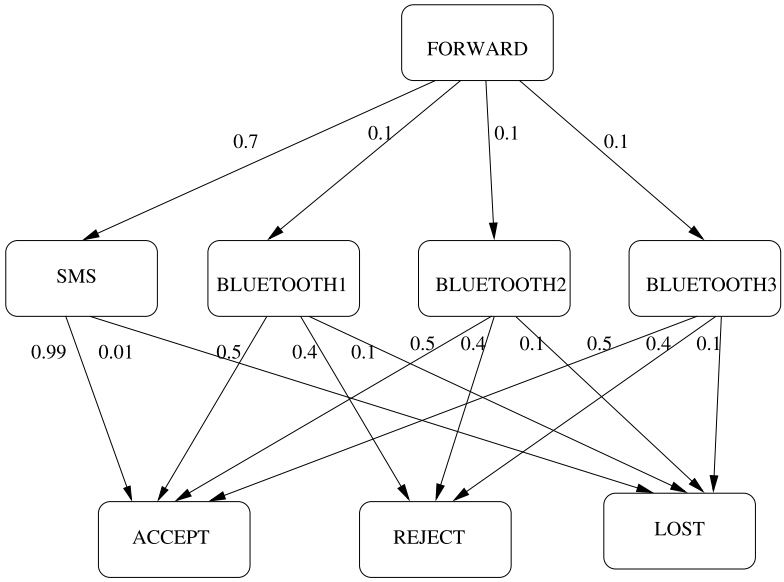


Fig. 3. A simple model of the message forwarding component of the Scatterbox system.

the other hand, when the system sends a message via SMS, the user does not have the option to reject the message, and the probability of the message arrival is 0.99 and message loss is 0.01. This is due to the fact that SMS technology is more reliable than Bluetooth, and the rate of loosing messages in transmission is lower.

Properties of probabilistic systems are specified in the logic PCTL [39]. We can model check some typical safety and liveness properties, such as "what is the probability, from the initial state of the model, of a message eventually being accepted or rejected?".

Although the simplified model of the message forwarding component can be used to verify some simple probabilistic properties, the model does *not* include many important characteristics of the behaviour of the pervasive message forwarding system. For example, the model should denote the *spatial* configurations of locations over time as well as the probabilistic transition of system states. We must therefore refine our model in order to construct a more realistic system model, and formalise more meaningful properties.

To model the evolution of spatial configurations, we use the notion of *location*. A location is a generic place where an execution occurs. It can be a mobile device, a lecture room, a sensor, a web page, a data object, etc. Locations can be nested within each other. For example, a location 'computer' may be contained in another location 'server'. Each location contains a number of 'local processes'. A location can enter/exist another location, etc. The details of such concepts can be found in [49].

In order to evaluate the behaviour of the message forwarding component more accurately and realistically, we refine the model in Fig. 3. The refined model, as shown in Fig. 4, includes both probabilistic behaviour and the evolution of spatial configurations of the locations of the Scatterbox system. Namely, the model contains probability information for transitions between different states, and the information how the spatial configuration changes over time. Each state in the model is represented as a labelled graph (see Fig. 4), where each node is represented by the location name and the state of its local processes. Note that the location names are *nominals*, i.e., reference names to locations, and the states of local processes are sets of atomic propositions. The edges represent the *sub-location* relation, where the child location is intuitively inside the parent location. As outlined in Fig. 4, the spatial configuration changes when the system states change in different transitions.

Before giving an intuitive description of the model, let us first give an intuition for the locations. The locations comprise the message, the Bluetooth sensors, the Scatterbox server, the SMS server and the user's mobile device. We use the following nominals as locations names. `SCATTERBOX` denotes the root, i.e., the outermost, location. Everything is in the domain (or location) `SCATTERBOX`. On the next level, we will see the Scatterbox server `SRV`, the Bluetooth sensors `B1, B2, B3`, the denoting the SMS server `SMS`, and the user's mobile device `MBL`. Finally, we consider a message `MSG` that is to be forwarded. We also define the following propositions: `accept1?` (resp. `accept2?, accept3?`), which denotes that `B1` (resp. `B2`, `B3`) has asked the user for the permission to deliver the message, `accept1` (resp. `accept2`, `accept3`), which denotes that the user has accepted the delivery of the message from `B1` (resp. `B2`, `B3`), `reject1` (resp. `reject2`, `reject3`), which denotes that the user has rejected the delivery of the message from `B1` (resp. `B2`, `B3`), `lost`, which denotes that the message has been lost, `in-range1` (resp. `in-range2, in-range3`), which denotes that the user's mobile device is in the range of `B1` (resp. `B2`, `B3`), and `no-bluet`, which denotes that the user's mobile device is not in the range of any of the Bluetooth sensors.

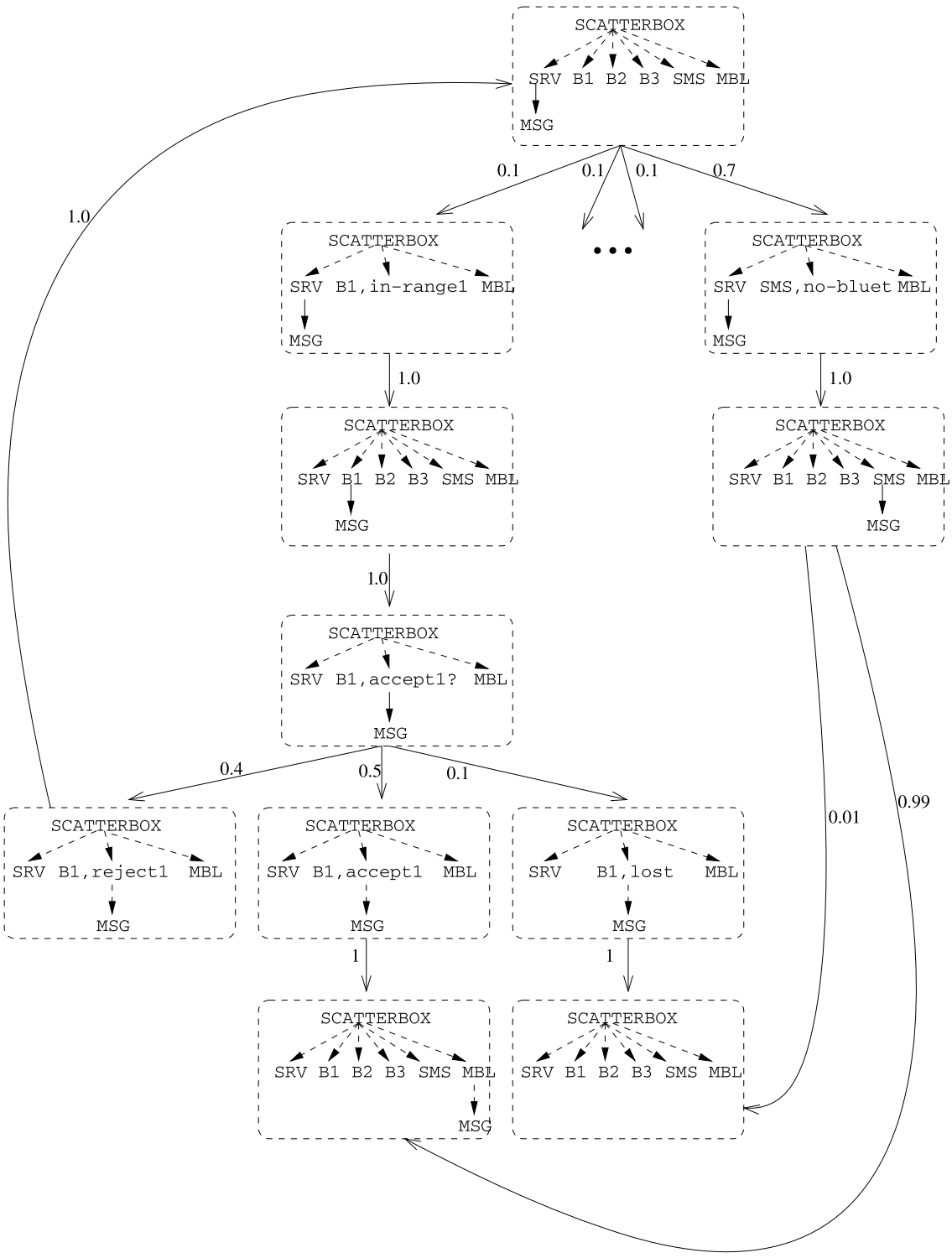


Fig. 4. A refined model of the message forwarding component illustrating the evolution of spatial configurations.

The model shown in Fig. 4 can be briefly described as follows. The message is initially in the Scatterbox server, denoted by the downward arrow from `SRV` to `MSG`. With a probability of 0.1, the user's mobile device is in the range of Bluetooth sensor 1, denoted by `B1,in-range1`. We recall that the proposition `in-range1` represents the state of the local process of `B1`. In this case, `B1` is in the range of the user's device. The Bluetooth sensors 2 and 3 can be treated similarly.

The message is then forwarded to the Bluetooth sensor 1, denoted by the downward arrow from B1 to MSG. This Bluetooth sensor then queries, whether or not the user will accept the delivery of the message, denoted by B1, accept1?. Subsequently, the message is rejected, accepted, or lost with a probability 0.4, 0.5, or 0.1, respectively.

From the initial state, there is also a probability of 0.7 that the user's mobile device is not in the range of any Bluetooth sensor, denoted by SMS,no-bluet.[7] The rest of the model follows the same pattern and can be read with the same intuition.

Based on the more realistic model of the message forwarding component given in Fig. 4, we now want to verify more complex properties. From a technical point of view, one may wish to express properties including explicit reference to individual locations. Such examples include:

- "a message cannot be delivered through a Bluetooth sensor without user's permission";
- "a message cannot be sent via a particular Bluetooth sensor if the user's mobile is not within the range of this sensor";
- "the probability that the server eventually becomes clear is greater than p";
- "B1 has at most one message at a time"; etc.

Clearly, we cannot express two dimensions simultaneously using a single formal framework. Thus, although PCTL can be used to specify the properties of the probabilistic behaviour, it cannot specify the properties regarding spatial dimension. A probabilistic model checker will not be sufficient to verify the properties mentioned above. We therefore need another technique for this more complex verification problem. In Section 6.3 we will solve the problem by combining the model checking methods of PCTL and Hybrid Logics.

In Figs. 3 and 4 we only considered the message forwarding behaviour of the Scatterbox system. If we also take the e-mail retrieval and context gathering behaviours into account, we will have a more complex model, which is shown in Fig. 5. The new model now has also a *real-time* dimension as well, which can be modelled with a timed automaton [50].

We can informally describe the model in Fig. 5 as follows: When a new e-mail arrives, the e-mail gathering component downloads it within t1 seconds. The system then asks for any new contextual information to decide whether to forward the message or not. At this point the system either forwards the message, cancels the message, or waits t2 seconds before it asks for new contextual information. In the WAIT state, if new contextual information does not arrive within t_max time units, the system cancels the message forwarding. If the contextual information arrives, and the reasoner decides not to forward the message, the message forwarding is cancelled. If the contextual information arrives and the reasoner decides to forward it, the message is forwarded to the message forwarding component, which is depicted in Fig. 5 by a transition from CONTEXT to MSG_FRWRD, where MSG_FRWRD denotes the model in Fig. 4.

We note that for the simplicity some of the location relations were not shown in Fig. 5. For example, in INIT, MSG is not within SRV; in RECEIVE, MSG moves into SRV; in WAIT, CNTX (denoting the context information) is not within SRV; in CONTEXT SRV contains CNTX; etc. Clearly, the model in Fig. 5 is not a homogeneous timed automaton. In fact, it is a combined model including real-time, probabilistic and spatial dimensions. Therefore, we can no longer use the standard verification techniques defined just for timed automata. A real-time temporal logic, such as TCTL [40], can be used to express the real-time dimension, but it cannot express the probabilistic and spatial dimensions. For example, some of the properties that we may wish to verify are as follows: "there exists a behaviour of the system that, within t seconds a message will be in the range of the Bluetooth sensor 1"; "when a message arrives, the probability of the server eventually being clear is greater than p"; "when a message arrives, the system will not wait for context information forever"; etc. As can be seen, these specifications tend to explore many dimensions simultaneously. Therefore, a single framework will not be sufficient to express the properties we are interested in without quite complex, and often contorted, encoding. In Section 6.3, we will show how to use our earlier combination technique to solve this problem by simply combining the logics TCTL, PCTL and Hybrid Logic.

6.3. Combining logics

In this section we first define a combined structure that pairs probabilistic and hybrid logics and allows us to capture the message forwarding behaviour depicted in Fig. 4, modelling both probabilistic transitions of system states and spatial distributions of locations. The probabilistic behaviour can be expressed in PCTL and the spatial behaviour can be expressed in Hybrid logic (HL) [46]. Thus, the behaviour of the message forwarding component can be modelled through a suitable combination of PCTL and Hybrid logics. For the case of Fig. 4, we can take this choice of combination just as 'temporalisation'.

Thus the message forwarding behaviour depicted in Fig. 4 can be modelled by means of a temporalised model $\langle S, T, g\rangle$, where $T = \langle \mathcal{E}, \mu\rangle$. The outer frame $\langle S, T\rangle$ models the probabilistic behaviour over time, and for every $s \in S$, $g(s)$ is a hybrid model which is a labelled rooted graph representing the topology of the structure of locations.

A 'temporalised' combined logic PCTL(HL) can be used to express the requirements of the message forwarding component, mentioned in Section 6.2. Probabilistic statements are captured by PCTL, while spatial statements are captured by HL.

[7] To allow for a more compact representation, we have associated the proposition no-bluet with the nominal SMS. Alternatively, we could have defined such a proposition for each Bluetooth sensor, similar to the not-in-range1 attribute.

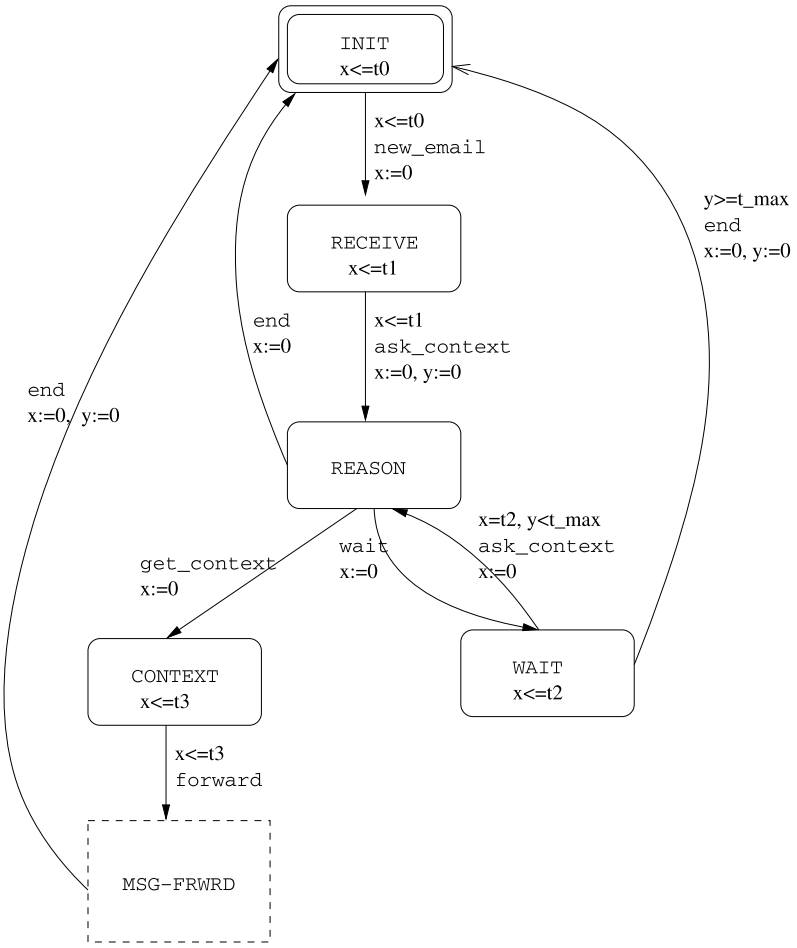


Fig. 5. The refined model including the e-mail retrieval, context gathering and message forwarding behaviour.

6.3.1. The constituent logics

PCTL is a probabilistic extension of CTL, as discussed in Section 3.2. HL is an extension of modal logic with a reference to locations, called *nominals*. Basically, HL contains nominals and the operators @ and $\downarrow$ as well as the standard existential modality $\Diamond$ and the universal modality $\Box$. Intuitively, $@_i\varphi$ holds if, and only if, φ holds at the location named by i, while $\downarrow x.\varphi$ holds if, and only if, φ holds whenever the current location is named by x.

6.3.2. Defining PCTL(HL)

Assume that φ_{HL} denotes *monolithic formulas* of HL. The language of PCTL(HL)[8] is defined by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_{\mathrm{HL}} \mid \mathrm{P}_{\sim r}[\bigcirc\varphi] \mid \mathrm{P}_{\sim r}[\varphi_1 \mathcal{U} \varphi_2]$$

In the following, the expressive power of the resulting combined logic is shown by specifying a number of meaningful properties of Scatterbox locations, mixing probabilistic and temporal requirements. Specifically, we consider the examples mentioned in Section 6.2.

The property that "the probability of the servers eventually being clear is greater than r" can expressed in PCTL(HL) as follows:

$$\mathrm{P}_{\geqslant r}[\mathtt{true}\,\mathcal{U} @_{\mathrm{SRV}} \neg\Diamond \mathrm{MSG}]$$

The property that "a message cannot be delivered through a Bluetooth sensor without the user's permission" is translated into PCTL(HL) as follows:

$$\mathrm{P}_{=0}[(@_{\mathrm{B1}} \neg \mathtt{accept1} \wedge @_{\mathrm{B2}} \neg \mathtt{accept2} \wedge @_{\mathrm{B3}} \neg \mathtt{accept3} \wedge @_{\mathrm{SMS}} \neg\Diamond \mathrm{MSG}) \mathcal{U} @_{\mathrm{MBL}} \Diamond \mathrm{MSG}]$$

[8] We do not include the bounded until operator within the syntax of PCTL.

The property "a message cannot be sent via a particular Bluetooth sensor if the user's mobile is not within the range of this sensor" is formally specified as follows:

$$\mathrm{P}_{=0}\big[(@_{\mathrm{B1}}\neg\texttt{in-range1}\,\mathcal{U}\,@_{\mathrm{B1}}\Diamond\mathtt{MSG}) \vee (@_{\mathrm{B2}}\neg\texttt{in-range2}\,\mathcal{U}\,@_{\mathrm{B2}}\Diamond\mathtt{MSG}) \vee (@_{\mathrm{B3}}\neg\texttt{in-range3}\,\mathcal{U}\,@_{\mathrm{B3}}\Diamond\mathtt{MSG})\big]$$

Note that none of the properties above can be expressed in either of PCTL or HL alone. As shown above, using the combined logic PCTL(HL) we can specify two dimensions at the same time.

A model checker for PCTL(HL) can be synthesised from component model checkers PCTL and HL. Using this combined model checker we can verify the properties expressed in the combined framework over the combined model in Fig. 4. We already proved that the combined model checking procedure transfers complexities from model checking within the component logics. We know that PCTL model checking over a DTMC has a polynomial complexity [39] and checking HL over hybrid model has PSPACE complexity. We can therefore show that the model checking problem for the combined logic PCTL(HL) has a PSPACE upper bound. This means that we can, in principle, model check the combined probabilistic and spatial dimensions at the same complexity of the spatial dimension itself.

6.3.3. Extending to TCTL(PCTL(HL))

We now present a framework combining real-time, probabilistic and spatial aspects to capture the e-mail retrieval, context gathering and message forwarding behaviours of the Scatterbox system, as depicted in Fig. 5. The combined structure in Fig. 5 can be modelled by means of a temporalised combination of a timed automaton and a PCTL(HL) model.

In order to express the real-time aspect, we consider a real-time extension of CTL, the logic TCTL. In TCTL temporal operators are equipped with timing constraints, used to express real-time properties. For example, the formula $\exists\Diamond^{\leqslant 2}\varphi$ asserts that there is a state satisfying φ, which can be reached within 2 seconds.

We can now define the 'temporalised' combined logic TCTL(PCTL(HL)) to express the requirements of the e-mail retrieval, context gathering and message forwarding behaviour of the Scatterbox system.

The language of TCTL(PCTL(HL)) is defined according to following rules:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_{\mathrm{PCTL(HL)}} \mid \exists\varphi_1\mathcal{U}^{\sim k}\varphi_2 \mid \forall\varphi_1\mathcal{U}^{\sim k}\varphi_2$$

where $\varphi_{\mathrm{PCTL(HL)}}$ denotes monolithic formulas of PCTL(HL). We now specify the properties mentioned in Section 6.2 using the combined framework:

The property that "there exists a behaviour of the system that within t seconds a message will be in the range of the Bluetooth sensor 1" can be expressed in TCTL(PCTL(HL)) as follows:

$$\exists\Diamond\big(\exists\Diamond^{\leqslant 50}@_{\mathrm{B1}}\,\texttt{in-range1}\big)$$

The property that "when a message arrives, the probability of the server's eventually being clear is greater than r" is translated into TCTL(PCTL(HL)) as follows:

$$\exists\Diamond\big(@_{\mathrm{SRV}}\Diamond\mathtt{MSG} \wedge \mathrm{P}_{\geqslant r}[\mathtt{true}\,\mathcal{U}@_{\mathrm{SRV}}\neg\Diamond\mathtt{MSG}]\big)$$

The property "when a message arrives, the system will not wait for a context information forever" is formally specified as follows:

$$\exists\Diamond\big(@_{\mathrm{SRV}}\Diamond\mathtt{MSG} \wedge @_{\mathrm{SRV}}\neg\Diamond\mathtt{CNTX} \wedge \exists\Diamond^{\leqslant t}@_{\mathrm{SRV}}\Diamond\mathtt{CNTX}\big)$$

where t is a constant.

Note that these specifications have several dimensions simultaneously. Therefore, a single framework will not be sufficient to concisely express these properties. On the other hand, TCTL(PCTL(HL)) allows us to have all these dimensions in a formula at the same time.

The complexity of model checking TCTL(PCTL(HL)) is the maximum complexity of model checking the component logics. We have already shown that the model checking complexity of PCTL(HL) is PSPACE. By Theorem 5.4, the combined logic TCTL(PCTL(HL)) has a PSPACE upper bound. This means that we can model check the real-time, probabilistic and spatial dimensions simultaneously in PSPACE.

6.4. Discussion

Pervasive systems are complex, and their formal description can involve many different logical dimensions. Therefore, properties that express the dynamics of a pervasive system frequently contain several aspects. In the Scatterbox system discussed in this section, for example, we needed at least three dimensions: nominals, probability, and real-time.

When using existing ad-hoc approaches to model checking, we can analyse only one of these dimensions in isolation. This is, however, not sufficient, because we cannot model check properties involving several dimensions, and we therefore cannot perform the required analysis of different interdependent system aspects.

Our combination method shows that one can often benefit from model checking, even in these more complex cases. On the Scatterbox system, we have applied our approach to particular logics relevant for this pervasive system. We have shown that a single logic is not sufficient to analyse complex properties, such as those describing the dynamics of the Scatterbox system, and that our combination approach can be used to express and analyse more complex properties that cannot be handled when using a single framework.

Note that we do not want to work on artificially created case studies. The requirement to simultaneously reason about different dimensions is natural in pervasive computing. In this article, we have therefore considered a genuine and realistic pervasive system as a case study.

7. Conclusion

In this article, we have presented a modular approach to model checking multi-agent systems. Instead of introducing new logics for combinations of different aspects and studying the resulting model checking problems, we have combined logics that represent different aspects and introduced a generic model checking method for different combinations of logics. In this way, many aspects of multi-agent systems, such as knowledge and time, knowledge and probability, real-time and knowledge, etc., can be viewed as simple standard combinations of logics and, without introducing a new logic, a generic combined model checking procedure can be synthesised from model checkers of simpler component logics. As we generally do *not* have verification tools for all the appropriate combinations, the application of the above approach can avoid the need for complex re-implementation of model checking procedures. For example, we might have verification tools for logics of knowledge, logics of context, real-time temporal logics, or probabilistic temporal logics, yet we have no tool that can verify a description containing *all* these dimensions.

We showed how our combined model checking method can be used to describe some existing combined approaches that have been applied in agent verification. There are numerous others that could be tackled, including alternating temporal logics, logics of intention, logics of belief, etc. We can also describe new, and as yet un-implemented, combinations. For example, we can combine the probabilistic and cooperation aspects, real-time and intention aspects, belief and uncertainty aspects, etc.

We showed that the generic model checking algorithm terminates, and is both sound and complete. We have also analysed the computational complexity of the resulting method, and have shown that the complexity of the synthesised model checker is essentially the supremum of the complexities of the component model checkers. This result confirms that modularity is easier to obtain in model checking than in deductive approaches, where combination often leads to exponential (or worse) complexity.

To illustrate and further motivate our approach—and to demonstrate its usefulness—we tackled a message forwarding system called *Scatterbox* and instantiated our approach to particular logics relevant for this pervasive system. We showed that a single model checking method is not a natural mechanism to verify complex systems such as Scatterbox, and that our combination method can solve more complex (and useful) verification problems, which cannot easily be handled within a single framework.

Finally, the future work here centres around implementing the technique and actually applying it in practise to a range of case studies. We will also consider logics in which region graphs are not enough to devise a model checking algorithm.

Acknowledgements

This work was partly funded by the Engineering and Physical Science Research Council (EPSRC) through the grants EP/F033567/1 “Verifying Interoperability Requirements in Pervasive Systems” and EP/H046623/1 “Synthesis and Verification in Markov Game Structures”. We would also like to thank Simon Dobson and Stephen Knox for providing the Scatterbox application and helpful discussions during the modelling of the system.

References

[1] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, MIT Press, 1999.
[2] A. Pnueli, The temporal logic of programs, in: Proc. 18th Symp. Foundations of Computer Science (FOCS), IEEE Computer Society, 1977, pp. 46–57.
[3] E.A. Emerson, Temporal and modal logic, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, 1990, pp. 996–1072.
[4] M. Kacprzak, A. Lomuscio, W. Penczek, Verification of multiagent systems via unbounded model checking, in: Proc. 3rd Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS), IEEE Computer Society, 2004, pp. 638–645.
[5] M. Wooldridge, Introduction to Multiagent Systems, John Wiley & Sons, Inc., 2001.
[6] R. van der Meyden, N.V. Shilov, Model checking knowledge and time in systems with perfect recall, in: Proc. Symp. Foundations of Software Technology and Theoretical Computer Science (FSTTCS), in: LNCS, vol. 1738, 1999, pp. 432–445.
[7] W. van der Hoek, M. Wooldridge, Model checking knowledge and time, in: Proc. 9th Int. SPIN Workshop on Model Checking of Software, Springer-Verlag, 2002, pp. 95–111.
[8] P. Gammie, R. van der Meyden, MCK: Model checking the logic of knowledge, in: Proc. Int. Conf. Computer Aided Verification (CAV), in: LNCS, vol. 3114, 2004, pp. 256–259.
[9] F. Raimondi, A. Lomuscio, A tool for specification and verification of epistemic properties in interpreted systems, Electronic Notes in Theoretical Computer Science 85 (2) (2004) 179–191.
[10] L. Wu, K. Su, Q. Chen, Model checking temporal logics of knowledge and its application in security verification, in: Computational Intelligence and Security, in: LNCS, vol. 3801, 2005, pp. 349–354.

[11] A. Lomuscio, F. Raimondi, Model checking knowledge, strategies, and games in multi-agent systems, in: Proc. 5th Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS), ACM, 2006, pp. 161–168.
[12] B. Konikowska, W. Penczek, Model checking for multivalued logic of knowledge and time, in: Proc. 5th Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS), ACM, 2006, pp. 169–176.
[13] K. Su, A. Sattar, X. Lu, Model checking temporal logics of knowledge via OBDDs, Computer Journal 50 (4) (2007) 403–420.
[14] A. Lomuscio, C. Pecheur, F. Raimondi, Automatic verification of knowledge and time with NuSMV, in: Proc. 20th Int. Joint Conf. on Artificial Intelligence (IJCAI), 2007, pp. 1384–1389.
[15] M. Bagic, A. Babac, M. Ciglaric, Verifying epistemic properties of multi-agent systems via action-based temporal logic, in: Proc. Conf. Comp. Intel. for Modelling, Control and Automation, 2008, pp. 470–475.
[16] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi, Reasoning About Knowledge, MIT Press, 1995.
[17] W. van der Hoek, M. Wooldridge, Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications, Studia Logica 75 (1) (2003) 125–157.
[18] L.A. Dennis, M. Fisher, M. Webster, R.H. Bordini, Model checking agent programming languages, Automated Software Engineering 19 (1) (2012) 5–63.
[19] Z. Cao, Model checking for real-time temporal, cooperation and epistemic properties, in: Proc. Intelligent Information Processing III, vol. 228, International Federation for Information Processing, 2007, pp. 63–72.
[20] A. Lomuscio, W.P.B. Woźna, Bounded model checking for knowledge and real-time, Artificial Intelligence 171 (16–17) (2007) 1011–1038.
[21] J.Y. Halpern, Reasoning About Uncertainty, MIT Press, 2003.
[22] N. de Carvalho Ferreira, M. Fisher, W. van der Hoek, Specifying and reasoning about uncertain agents, International Journal of Approximate Reasoning 49 (1) (2008) 35–51.
[23] C. Delgado, M. Benevides, Verification of epistemic properties in probabilistic multi-agent systems, in: Proc. Conf. Multiagent System Technologies (MATES), in: LNCS, vol. 5774, 2009, pp. 16–28.
[24] A. Hinton, M.Z. Kwiatkowska, G. Norman, D. Parker, PRISM: A tool for automatic verification of probabilistic systems, in: Proc. TACAS'06, in: LNCS, vol. 3920, Springer, 2006, pp. 441–444.
[25] M. Franceschet, A. Montanari, M. de Rijke, Model checking for combined logics with an application to mobile systems, Automated Software Engineering 11 (3) (2004) 289–321.
[26] W. Penczek, A. Lomuscio, Verifying epistemic properties of multi-agent systems via bounded model checking, Fundamenta Informaticae 55 (2) (2002) 167–185.
[27] M. Finger, D.M. Gabbay, Combining temporal logic systems, Notre Dame Journal of Formal Logic 37 (2) (1996) 204–232.
[28] B. Bennett, C. Dixon, M. Fisher, E. Franconi, I. Horrocks, M. de Rijke, Combinations of modal logics, AI Review 17 (1) (2002) 1–20.
[29] D. Gabbay, A. Kurucz, F. Wolter, M. Zakharyaschev, Many-dimensional modal logics: Theory and applications, in: Studies in Logic and the Foundations of Mathematics, vol. 148, Elsevier Science, 2003.
[30] A. Kurucz, Combining modal logics, in: J. van Benthem, P. Blackburn, F. Wolter (Eds.), Handbook of Modal Logic, in: Studies in Logic and Practical Reasoning, vol. 3, Elsevier, 2007, pp. 869–924.
[31] M. Finger, D.M. Gabbay, Adding a temporal dimension to a logic system, Journal of Logic, Language, and Information 1 (1992) 203–234.
[32] P. Blackburn, M. de Rijke, Why combine logics? Studia Logica 59 (1997) 5–27.
[33] E. Hemaspaandra, Complexity transfer for modal logic, in: Proceedings of Symposium on Logic in Computer Science, 1994, pp. 164–173.
[34] F. Wolter, Fusions of modal logics revisited, in: Proc. Advances in Modal Logic, in: CSLI Lecture Notes, 1998, pp. 361–379.
[35] F. Wolter, Completeness and decidability of tense logics closely related to logics above K4, Journal of Symbolic Logic 62 (1) (1997) 131–158.
[36] M. Marx, Complexity of products of modal logics, Journal of Logic and Computation 9 (1999) 221–238.
[37] A. Kurucz, S5 × S5 × S5 lacks the finite model property, in: Advances in Modal Logic, 2000, pp. 321–327.
[38] F. Wolter, The product of converse PDL and polymodal K, Journal of Logic and Computation 10 (2) (2000) 223–251.
[39] H. Hansson, B. Jonsson, A logic for reasoning about time and reliability, Formal Aspects of Computing 6 (1994) 102–111.
[40] R. Alur, C. Courcoubetis, D. Dill, Model-checking in dense real-time, Information and Computation 104 (1993) 2–34.
[41] C. Baier, J.-P. Katoen, Principles of Model Checking (Representation and Mind Series), MIT Press, 2008.
[42] J. Sproston, Strict divergence for probabilistic timed automata, in: Proceedings of the 20th International Conference on Concurrency Theory, CONCUR 2009, Springer-Verlag, 2009, pp. 620–636.
[43] M. Kwiatkowska, G. Norman, R. Segala, J. Sproston, Automatic verification of real-time systems with discrete probability distributions, Theoretical Computer Science 282 (2002) 101–150.
[44] A. Troina, Probabilistic timed automata for security analysis and design, PhD thesis, Dipartimento di Informatica, Universita degli Studi di Pisa, 2006.
[45] A. Bianco, L.D. Alfaro, Model checking of probabilistic and nondeterministic systems, in: Proc. Symp. Foundations of Software Technology and Theoretical Computer Science (FSTTCS), in: LNCS, vol. 1026, 1995, pp. 499–513.
[46] C. Areces, Hybrid logics: The old and the new, in: Proceedings of LogKCA-07, 2007, pp. 15–29.
[47] S. Knox, R. Shannon, L. Coyle, A. Clear, S. Dobson, A. Quigley, P. Nixon, Scatterbox: Context-aware message management, Revue d'Intelligence Artificielle 22 (5) (2008) 549–568.
[48] S. Konur, M. Fisher, S. Dobson, S. Knox, Formal verification of a pervasive messaging system, Formal Aspects of Computing (2013), http://dx.doi.org/10.1007/s00165-013-0277-4, in press.
[49] L. Cardelli, A.D. Gordon, Mobile ambients, Theoretical Computer Science 240 (1) (2000) 177–213.
[50] R. Alur, Timed automata, Theoretical Computer Science 126 (1999) 183–235.