Mind the Gap! Runtime Verification of Partially Observable MASs with Probabilistic Trace Expressions

Davide Ancona, Angelo Ferrando, and Viviana Mascardi(✉)

University of Genova, Via Dodecaneso 35, 16146 Genova, Italy
{davide.ancona,angelo.ferrando,viviana.mascardi}@unige.it

Abstract. In this paper we present the theory behind Probabilistic Trace Expressions (PTEs), an extension of Trace Expressions where types of events that can be observed by a monitor are associated with an observation probability. PTEs can be exploited for monitoring that agents in a MAS interact in compliance with an Agent Interaction Protocol (AIP) modeled as a PTE, even when the monitor realizes that an interaction took place in the MAS, but it was not correctly observed ("observation gap"). To this aim, we adapt an existing approach for runtime verification with state estimation, we present a semantics for PTEs that allows for the estimation of the probability to reach a given state, given a sequence of observations which may include observation gaps, we present a centralized implemented algorithm to dynamically verify the behavior of the MAS under monitoring and we discuss its potential and limitations.

Keywords: Probabilistic Trace Expressions · Partial observability · State estimation · Multiagent systems · Agent interaction protocols

1 Introduction

Runtime verification of complex, distributed systems under ideal conditions (perfect observability of all the relevant events, no leaky communication channels, etc.) is an hard task to perform, and has been addressed by many scientific works including surveys and introductory papers [14,24,27], books [13], seminars [18,23], and conferences[1]. When the conditions are not ideal and some relevant events cannot be observed by the monitor, generating a *gap* in the event trace, the problem becomes even harder [11,15,22,25,31]. A gap represents the absence of information in the analyzed trace and corresponds to an execution point – or to a time slot – where the monitor does not know what the system did. Gaps may be due to the process of sampling observed events to reduce monitoring overhead, but also to events that are partially observable or not observable at all by the monitor: the monitor might be aware that an event took place, but does not know which. We say that the monitor "observes a gap" to describe this

[1] http://www.runtime-verification.org/.

D. Baumeister and J. Rothe (Eds.): EUMAS 2022, LNAI 13442, pp. 22–40, 2022.
https://doi.org/10.1007/978-3-031-20614-6_2

situation. The introduction of gaps raises problems in checking that a temporal property is verified by the system, given that a trace of events (which may include gaps) has been observed. If the monitor does not know which event has been observed, it cannot know whether the temporal property is satisfied or not.

In [32], each time a gap in observed a Hidden Markov Model (HMM) of the system is queried to know which events could be observed in the current state of the system, and with which probability. This allows the authors to estimate the probability to reach some state s_i after observing $obs = O_1, O_2, ..., O_t$ events, and – by generating a monitor that combines the system HMM and the temporal property ϕ into a single integrated model – to estimate the probability that ϕ is satisfied after observing $obs = O_1, O_2, ..., O_t$ events.

In this paper, we take [32] as our starting point, and we combine the approach presented therein with the adoption of an existing expressive formalism to model systems and properties, Trace Expressions [1,2,5,6,10].

After an overview of the background in Sect. 2, we present Probabilistic Trace Expressions (PTEs) which extend Trace Expressions with probabilities associated with event types (Sect. 3). PTEs are more expressive than HMM, deterministic finite state machines and linear time temporal logic (LTL [28]), being able to model more than context free languages. In Sect. 4 (1) we use PTEs to model the probabilistic behaviour of the system under observation, possibly starting from an HMM and then refining or extending it; (2) we show how – by applying the rules defining the operational semantics of PTEs – we obtain the same results of the forward algorithm presented in [32]; (3) we present the *Probabilize* algorithm for transforming Trace Expressions corresponding to LTL properties into PTEs; (4) by joining the two representations obtained in steps 1 and 3 above using the $\wedge$ conjunction operator natively provided by PTEs, we obtain for free a way to verify satisfaction of LTL properties in presence of observation gaps. Section 5 discusses the implementation of a centralized algorithm for Runtime Verification of partially observable MASs and suggests that a decentralized approach may solve some of its limitations, at the expense of communication overhead among the monitors. Future directions of our research are addressed in Sect. 6.

2 Background

Hidden Markov Models. A Hidden Markov Model (HMM [16,30]) is a statistical Markov model where the system being modeled is assumed to be a Markov process with hidden states. It can be modeled as a quintuple $H = \langle S, A, V, B, \Pi \rangle$ where

- $S = \{s_1, ..., s_{N_s}\}$ is the set of states;
- A is the $N_s \times N_s$ transition probability matrix: $A_{i,j} = \Pr(\text{state is } s_j \text{ at time } t+1 \mid \text{state is } s_i \text{ at time } t)$;
- $V = \{v_1, ..., v_{N_v}\}$ is the set of observation symbols;
- B is the $N_s \times N_v$ observation probability matrix: $B_{i,j}$, also denoted with $b_i(v_j)$ for clarity, is $\Pr(v_j \text{ is observed at time } t \mid \text{state is } s_i \text{ at time } t)$;

– $\Pi = \{\pi_1, ..., \pi_{N_s}\}$ is the initial state distribution: π_i is the probability that the initial state is s_i.

We use as our running example the one presented in [32], where a model of a planetary rover mission is modeled. The rover hosts two generic instruments, A and B, and all the events generated by the rover are recorded on a log file. We consider four different kinds of events, inspired by Barringer et al. [12]:

– *command* (`cmd` in the HMM figure), the command submitted to the rover;
– *dispatch* (`disp`), the dispatch of the command from the rover to the instrument;
– *success* (`succ`), the success of the command on the instrument;
– *fail* (`fail`), the failure of the command on the instrument.

All these events are characterized by three parameters: the instrument id (a or b), the issued command (*start* or *reset*), and a time stamp indicating when the event occurred. When the rover receives a command, it reports the information to the logger and sends the command to the relevant instrument. Once received the command, the instrument issues a dispatch event to the logger and then executes the command. If the execution is successful (resp. fails), a corresponding success (resp. failure) event is reported to the logger. It is also possible that the command is simply lost for some reason and neither a success nor a fail occurs. Events have some probability to be observed, and the chance to move from one state to another is also modeled by a probability.

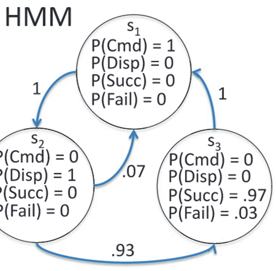

Fig. 1. An example of HMM (from [32]).

Figure 1 represents an HMM inspired to the rover example, where

– $S = \{s_1, s_2, s_3\}$;
– $A_{1,1} = A_{1,3} = 0; A_{1,2} = 1;\ A_{2,1} = 0.07; A_{2,2} = 0; A_{2,3} = 0.93;$ $A_{3,1} = 1; A_{3,2} = A_{3,3} = 0;$
– $V = \{C, D, S, F\}$ (C stands for `cmd`, D for `disp`, etc.);
– $b_1(C) = 1; b_1(D) = b_1(S) = b_1(F) = 0;\ b_2(D) = 1; b_2(C) = b_2(S) = b_2(F) = 0;\ b_3(C) = b_3(D) = 0; b_3(S) = 0.97; b_3(F) = 0.03;$
– $\pi_1 = 1, \pi_2 = \pi_3 = 0$ (not shown in the figure).

To compute the probability that an HMM H ends in a specific state given an observation sequence $O = \langle O_1, O_2, ..., O_T \rangle$, the forward algorithm can be used [29]. Let $Q = \langle q_1, q_2, ..., q_T \rangle$ denote the (unknown) state sequence that the system passed through, i.e., q_t denotes the state of the system when observation O_t is made. Let $\alpha_t(i) = Pr(O_1, O_2, ..., O_t, q_t = s_i | H)$, i.e., the probability that the first t observations yield $O_1, O_2, ..., O_t$ and that q_t is s_i, given the model H. The base case is:

$$\alpha_1(j) = \pi_j b_j(O_1) \text{ for } 1 \leq j \leq N_s$$

whereas the recursive case is:

$$\alpha_{t+1}(j) = (\Sigma_{i=1..N_s} \alpha_t(i) A_{i,j}) b_j(O_{t+1}) \text{ for } 1 \leq t \leq T-1 \text{ and } 1 \leq j \leq N_s$$

Trace Expressions. Trace expressions are based on the notions of *event* and *event type*. $\mathcal{E}$ denotes the fixed universe of events subject to monitoring. An event trace over $\mathcal{E}$ is a possibly infinite sequence of events in $\mathcal{E}$, and a Trace Expression over $\mathcal{E}$ denotes a set of event traces over $\mathcal{E}$. Trace expressions are built on top of event types (chosen from a set $\mathcal{ET}$), each specifying a subset of events in $\mathcal{E}$. A Trace Expression $\tau \in \mathcal{T}$ represents a set of possibly infinite event traces, and is defined on top of the following operators:

- ϵ (empty trace), denoting the singleton set $\{\epsilon\}$ containing the empty event trace ϵ.
- $\vartheta{:}\tau$ (*prefix*), denoting the set of all traces whose first event e matches the event type ϑ, and the remaining part is a trace of τ.
- $\tau_1{\cdot}\tau_2$ (*concatenation*), denoting the set of all traces obtained by concatenating the traces of τ_1 with those of τ_2.
- $\tau_1{\wedge}\tau_2$ (*intersection*), denoting the intersection of the traces of τ_1 and τ_2.
- $\tau_1{\vee}\tau_2$ (*union*), denoting the union of the traces of τ_1 and τ_2.
- $\tau_1|\tau_2$ (*shuffle*), denoting the set obtained by shuffling the traces of τ_1 with the traces of τ_2.

The derived constant Trace Expression 1 is equivalent to the expression $\tau = \epsilon{\vee}everyEvent{:}\tau$, where $everyEvent = \mathcal{E}$. Trace expressions support recursion through cyclic terms expressed by finite sets of recursive syntactic equations, as supported by modern Prolog systems. The semantics of Trace Expressions is specified by a transition relation $\delta \subseteq \mathcal{T} \times \mathcal{E} \times \mathcal{T}$, where $\mathcal{T}$ and $\mathcal{E}$ denote the set of Trace Expressions and of events, respectively. $\tau_1 \xrightarrow{e} \tau_2$ means $(\tau_1, e, \tau_2) \in \delta$; the transition $\tau_1 \xrightarrow{e} \tau_2$ expresses the property that the system under monitoring can safely move from the state specified by τ_1 into the state specified by τ_2 when event e is observed. A Trace Expression models the current state of a protocol. Protocol state transitions are ruled by the transition system shown in Fig. 2, which define δ.

Runtime Verification with State Estimation. Given a trace (possibly with gaps), in [32] Stoller et al. propose an approach to compute the probability that a LTL temporal property ϕ is satisfied by a system modeled by an HMM H, given that $obs = O_1, O_2, ..., O_t$ have been observed. More formally, they evaluate $Pr(\phi|\ obs, H)$ by applying the following steps:

$$\text{(prefix)}\,\frac{}{\vartheta{:}\tau \xrightarrow{e} \tau}\; e\in\vartheta \qquad \text{(or-l)}\,\frac{\tau_1 \rightarrow \tau_1}{\tau_1 \vee \tau_2 \xrightarrow{e} \tau_1'} \qquad \text{(or-r)}\,\frac{\tau_2 \rightarrow \tau_2}{\tau_1 \vee \tau_2 \xrightarrow{e} \tau_2'}$$

$$\text{(and)}\,\frac{\tau_1 \xrightarrow{e} \tau_1' \quad \tau_2 \xrightarrow{e} \tau_2'}{\tau_1 \wedge \tau_2 \xrightarrow{e} \tau_1' \wedge \tau_2'} \qquad \text{(shuffle-l)}\,\frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1 | \tau_2 \xrightarrow{e} \tau_1' | \tau_2} \qquad \text{(shuffle-r)}\,\frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1 | \tau_2 \xrightarrow{e} \tau_1 | \tau_2'}$$

$$\text{(cat-l)}\,\frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1 \cdot \tau_2 \xrightarrow{e} \tau_1' \cdot \tau_2} \qquad \text{(cat-r)}\,\frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1 \cdot \tau_2 \xrightarrow{e} \tau_2'}\; \epsilon(\tau_1)$$

$$\text{(ϵ-empty)}\,\frac{}{\epsilon(\epsilon)} \qquad \text{(ϵ-or-l)}\,\frac{\epsilon(\tau_1)}{\epsilon(\tau_1 \vee \tau_2)} \qquad \text{(ϵ-or-r)}\,\frac{\epsilon(\tau_2)}{\epsilon(\tau_1 \vee \tau_2)} \qquad \text{(ϵ-others)}\,\frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1 \, op \, \tau_2)}\; op\in\{|,\cdot,\wedge\}$$

Fig. 2. Transition system for trace expressions.

1. learn the HMM H from a given set of traces without gaps, using standard HMM learning algorithm;
2. generate the deterministic finite state machine (DFSM) corresponding to ϕ;
3. generate a monitor combining H and the DFSM to check the sequence *obs*.

Step 1 falls outside the boundaries of their investigation, and in the sequel we will disregard how the HMM has been created as well.

3 Probabilistic Trace Expressions

A probabilistic Trace Expression (PTE) is a Trace Expression where occurrences of event types in the expression have a probability associated with them. The probability is written after the occurrence of the event type, in square brackets. From a syntactic point of view, this extension is the only difference w.r.t. "normal" Trace Expressions introduced in Sect. 2.

Example. We present the PTE corresponding to the rover example. Event type *cmd* is { command(Inst, Comm, TS) such that Inst ∈ {a, b}, Comm ∈ {start, reset}, TS a time stamp in the range 0...3 }; event type *disp* is { dispatch(Inst, Comm, TS) }, *succ* = { success(Inst, Comm, TS) } and *fail* is { fail(Inst, Comm, TS) }. The resulting Trace Expression can be written in two equivalent (from the PTE semantics viewpoint) ways:

$$\tau_{s_1} = cmd[1]{:}\tau_{s_2}$$

$$\tau_{s_2} = disp[0.07]{:}\tau_{s_1} \vee disp[0.93]{:}\tau_{s_3}$$

$$\tau_{s_3} = succ[0.97]{:}\tau_{s_1} \vee fail[0.03]{:}\tau_{s_1}$$

(note the *disp* occurrence in both branches of τ_{s_2} definition, with different probabilities and different Trace Expressions after the ":" operator) and

$$\tau'_{init} = cmd[1]{:}\tau'_{s_2}$$

$$\tau'_{s_1} = \mathit{cmd}[0.07]{:}\tau'_{s_2}$$
$$\tau'_{s_2} = \mathit{disp}[1]{:}(\tau'_{s_1} \vee \tau'_{s_3})$$
$$\tau'_{s_3} = \mathit{succ}[0.9021]{:}\tau'_{s_1} \vee \mathit{fail}[0.0279]{:}\tau'_{s_1}$$

The Trace Expression in the first form tells us, for example, that the probability of the protocol to reach τ_{s_1} starting from τ_{s_2} and having observed *disp* is 0.07 while the probability to reach τ_{s_3} starting from τ_{s_2} and having observed *disp* is 0.93 (second equation of the first formulation). To make this information explicit, the transition from state s_2 to states s_1 and s_3 in the HMM has been modeled by $\tau_{s_2} = \mathit{disp}[0.07]{:}\tau_{s_1} \vee \mathit{disp}[0.93]{:}\tau_{s_3}$, introducing non-determinism due to the occurrence of the same event type *disp* in both branches of the "or" operator. While in a non probabilistic setting $\tau_{s_2} = \mathit{disp}{:}\tau_{s_1} \vee \mathit{disp}{:}\tau_{s_3}$ would be equivalent to $\tau_{s_2} = \mathit{disp}{:}(\tau_{s_1} \vee \tau_{s_3})$ and the second version would be definitely preferred, as – besides being more readable and compact – is deterministic, in a probabilistic setting this simplification would cause us to lose precious information on the probability to move to some state S, given some observed event O.

The second version overcomes this problem by propagating – via multiplication – the different probabilities associated with *disp* in ${s_2}'$ to the states ${s_1}'$ and ${s_3}'$ that can be reached from ${s_2}'$ (second and fourth equation of the second formulation). With this second form, we gain determinism at the price of adding an initial state τ_{init} for each state whose initial probability is not zero, and of losing the one-to-one clear correspondence with the HMM. As an example, in the fourth equation, understanding that *succ*[0.9021] comes from the probability 0.97 associated with observing *succ* in state ${s_3}'$ multiplied by the probability 0.93 of having reached ${s_3}'$ from ${s_2}'$ is far from intuitive.

Given that a structure-driven transformation from the first form to the second can be implemented in time linear with the Trace Expression length, we adopt the first form for presentation purposes, since it is closer to the HMM, but we use the second one in the implementation, since it is more efficient.

Like a "normal" Trace Expression, a PTE τ can be seen as the current state of a protocol that started in some initial state τ_{init} and reached τ after n events $O_1 ... O_n$ took place, that moved τ_{init} to τ through intermediate states τ_{q1}, τ_{q2}, ... , $\tau_{qn} = \tau$. If we denote with $\tau \overset{O}{\rightarrow} \tau'$ the transition from state τ to state τ' due to the event O taking place and being observed, we may write

$\tau_{init} \overset{O_1}{\rightarrow} \tau_{q1} \overset{O_2}{\rightarrow} \tau_{q2} \overset{O_3}{\rightarrow} \tau_{q3} ... \overset{O_n}{\rightarrow} \tau_{qn}$, where $\tau_{qn} = \tau$.

In order to properly manage probabilities, it is convenient to associate with τ – in an explicit and easily computable way – the probability of the protocol to have reached τ starting from τ_{init} and having observed $O_1 ... O_n$.

We define a "PTE state" (simply "state" from now on) the triple consisting of a Trace Expression τ, a sequence of events $O_1 ... O_n$ observed before reaching τ, and the probability π_τ that the protocol reached τ. We represent the state with the notation $\langle \tau, \pi_\tau, O_1 ... O_n \rangle$.

In this work, we are interested in analyzing the protocol evolution in presence of observation gaps: in a state τ_c (for $\tau_{current}$), the monitor driven by a PTE may either observe an event O, and then its behaviour is the same as in the non-probabilistic setting – it moves to the next state τ, if $\tau_c \xrightarrow{O} \tau$ is an allowed move –, or "observe a gap". Observing, or perceiving, a gap means that the monitor is aware that some event took place and hence the protocol must move one step forward, but it is also aware that the event has not been correctly observed. The monitor cannot commit to the $\tau_c \xrightarrow{O} \tau$ move in this case, but it must remember that many moves were possible, one for each of the events that could have taken place in τ, and that could have filled the perceived gap: $\tau_c \xrightarrow{gap} \tau$ (if the event were O, modeled by $gap(O)$ in the sequence of observed events), $\tau_c \xrightarrow{gap} \tau'$ (if the event were O', modeled by $gap(O')$), $\tau_c \xrightarrow{gap} \tau''$ (if the event were O'', modeled by $gap(O'')$), etc.

$$\text{(prefix)}\frac{}{\langle\vartheta[\pi_e]{:}\tau, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau, \pi_e * \pi_{tr}, obs\ \ any(e)\rangle}\ e\in\vartheta$$

$$\text{(or-l)}\frac{\langle\tau_1, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1', \pi_{tr}', obs\ \ any(e)\rangle}{\langle\tau_1\vee\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1', \pi_{tr}', obs\ \ any(e)\rangle}$$

$$\text{(or-r)}\frac{\langle\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_2', \pi_{tr2}, obs\ \ any(e)\rangle}{\langle\tau_1\vee\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_2', \pi_{tr2}, obs\ \ any(e)\rangle}$$

$$\text{(and)}\frac{\langle\tau_1, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1', \pi_{t1}, obs\ \ any(e)\rangle\ \ \langle\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_2', \pi_{t2}, obs\ \ any(e)\rangle}{\langle\tau_1\wedge\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1'\wedge\tau_2', \pi_{tr}', obs\ \ any(e)\rangle}\ \pi_{tr}'{=}f(\pi_{t1},\pi_{t2})$$

$$\text{(shuffle-l)}\frac{\langle\tau_1, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1', \pi_{tr}', obs\ \ any(e)\rangle}{\langle\tau_1|\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1'|\tau_2, \pi_{tr}', obs\ \ any(e)\rangle}$$

$$\text{(shuffle-r)}\frac{\langle\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_2', \pi_{tr}', obs\ \ any(e)\rangle}{\langle\tau_1|\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1|\tau_2', \pi_{tr}', obs\ \ any(e)\rangle}$$

$$\text{(cat-l)}\frac{\langle\tau_1, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1', \pi_{tr}', obs\rangle}{\langle\tau_1{\cdot}\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_1'{\cdot}\tau_2, \pi_{tr}', obs\ \ any(e)\rangle}$$

$$\text{(cat-r)}\frac{\langle\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_2', \pi_{tr}', obs\ \ any(e)\rangle}{\langle\tau_1{\cdot}\tau_2, \pi_{tr}, obs\rangle \xrightarrow{any} \langle\tau_2', \pi_{tr}', obs\ \ any(e)\rangle}\ \epsilon(\tau_1)$$

Fig. 3. Transition system for probabilistic trace expressions states.

The transition rules between states are shown in Fig. 3 and follow the pattern of the rules defined for Trace Expressions, with modifications for taking care of the probability propagation and of observed events including gaps. The rules for ϵ are the same as for normal Trace Expressions. Appendix A of the longer version of this paper available as a DIBRIS technical report provides a deep explanation of each of them [8].

In Fig. 3 the use of *any* and *any*(e) allows us to model the transition in the case that an event has been observed and in the case an observation gap took place, using the same rule. In fact, $\mathit{any} \in \{e, \mathit{gap}\}$ and if $\mathit{any} == e$ then $\mathit{any}(e) == e$; if $\mathit{any} == \mathit{gap}$ then $\mathit{any}(e) == \mathit{gap}(e)$.

If $\mathit{any} == e$, then e has been observed, the arrow modeling the state transition function $\xrightarrow{\mathit{any}}$ is actually labeled with e, and e is concatenated with the previously observed events, *obs*; if $\mathit{any} == \mathit{gap}$, then a gap took place, the arrow $\xrightarrow{\mathit{any}}$ is labeled with *gap*, and $\mathit{gap}(e)$, meaning that a gap took place, and that it could be filled with event e, is concatenated with the previously observed events.

Nondeterminism in State Transitions. The state transition function $\xrightarrow{\mathit{any}}$ is nondeterministic: one state can move into more than one state for many different reasons. Let us consider the *cmd* event type introduced at the beginning of this section. The transitions below can take place starting from the state $\langle \mathit{cmd}[0.3]{:}\tau, 0.2, \mathit{obs}\rangle$ when an observation gap occurs.

- $\langle \mathit{cmd}[0.3]{:}\tau, 0.2, \mathit{obs}\rangle \xrightarrow{\mathit{gap}} \langle \tau, 0.06, \mathit{obs}\ \mathit{gap}(\mathit{command}(a, \mathit{start}, 0))\rangle$
- $\langle \mathit{cmd}[0.3]{:}\tau, 0.2, \mathit{obs}\rangle \xrightarrow{\mathit{gap}} \langle \tau, 0.06, \mathit{obs}\ \mathit{gap}(\mathit{command}(b, \mathit{start}, 0))\rangle$
- ... plus 14 more transitions.

As another example, let us consider again the event type *cmd* defined above and the state $\langle \mathit{cmd}[0.75]{:}\tau_1 \vee \mathit{cmd}[0.25]{:}\tau_2, 0.4, \mathit{obs}\rangle$. If $\mathit{command}(a, \mathit{start}, 3)$ (abbreviated in $c(a, s, 3)$ for presentation purposes) is observed, both branches of the choice in $\mathit{cmd}[0.75]{:}\tau_1 \vee \mathit{cmd}[0.25]{:}\tau_2$ are valid, leading to the two transitions below.

- $\langle \mathit{cmd}[0.75]{:}\tau_1 \vee \mathit{cmd}[0.25]{:}\tau_2, 0.4, \mathit{obs}\rangle \xrightarrow{c(a,s,3)} \langle \tau_1, 0.3, \mathit{obs}\ c(a, s, 3)\rangle$
- $\langle \mathit{cmd}[0.75]{:}\tau_1 \vee \mathit{cmd}[0.25]{:}\tau_2, 0.4, \mathit{obs}\rangle \xrightarrow{c(a,s,3)} \langle \tau_2, 0.1, \mathit{obs}\ c(a, s, 3)\rangle$

If, starting from $\langle \mathit{cmd}[0.75]{:}\tau_1 \vee \mathit{cmd}[0.25]{:}\tau_2, 0.4, \mathit{obs}\rangle$, a gap is observed, the two sources of nondeterminism (the first due to the gap that can be filled with many events matching the expected event type, and the second due to the nondeterministic choice in the Trace Expression) combine together, generating 32 possible transitions. Other sources of nondeterminism in the Trace Expression are due to the shuffle and the concatenation operators, defined by two transitions rules each. Figure 4 presents the rules for dealing with nondeterminism and for introducing the notion of transitive closure of transitions:

(state-to-set) The function represented by $\rightarrow_\gamma$ takes one PTE state γ, one observed event or gap *any*, and returns the set of all the PTE states that γ can reach via $\xrightarrow{\mathit{any}}$.

(set-to-set) The function represented by $\twoheadrightarrow$ takes one set of PTE states $\{\gamma_1, \gamma_2, ..., \gamma_n\}$, one observed event or gap *any*, and returns the union of the sets of PTE states that each $\gamma_i \in \{\gamma_1, \gamma_2, ..., \gamma_n\}$ can reach via $\xrightarrow{\mathit{any}}_\gamma$.

$$\text{(state-to-set)}\frac{\gamma \overset{any}{\to} \gamma_1 \quad \gamma \overset{any}{\to} \gamma_2 \quad ... \quad \gamma \overset{any}{\to} \gamma_n}{\gamma \overset{any}{\to}_\gamma \{\gamma_1, \gamma_2, ..., \gamma_n\}}$$

$$\text{(set-to-set)}\frac{\gamma_1 \overset{any}{\to}_\gamma \Gamma_1 \quad \gamma_M \overset{any}{\to}_\gamma \Gamma_2 \quad ... \quad \gamma_n \overset{any}{\to}_\gamma \Gamma_n}{\{\gamma_1, \gamma_2, ..., \gamma_n\} \overset{any}{\twoheadrightarrow} \Gamma_1 \cup \Gamma_2 \cup ... \cup \Gamma_1}$$

$$\text{(closure)}\frac{\Gamma_0 \overset{O_1}{\twoheadrightarrow} \Gamma_1 \overset{O_2}{\twoheadrightarrow} \quad ... \quad \Gamma_{n-1} \overset{O_n}{\twoheadrightarrow} \Gamma_n}{\Gamma_0 \overset{O_1 ... O_n}{\twoheadrightarrow} \Gamma_n}$$

$$\text{(closure-init)}\frac{\{\langle\tau, 1, \sigma\rangle\} \overset{O_1 ... O_n}{\twoheadrightarrow} \Gamma_n}{\tau \overset{O_1 ... O_n}{\twoheadrightarrow} \Gamma_n} \ \sigma\text{=empty sequence}$$

Fig. 4. Rules for nondeterminism and transitive closure.

(closure) We use $\twoheadrightarrow$ to denote the transitive closure of $\twoheadrightarrow$ by putting the sequence of observed events on top of the arrow.

(closure-init) Finally, a PTE τ can evolve into any state $\gamma \in \Gamma_n$ after observation of $O_1 ... O_n$, if the PTE state $\langle\tau, 1, \sigma\rangle$ can, where σ is the empty sequence.

Example. Starting from the PTE τ_{s_1} used as running example, we have:

$$\tau_{s_1} \overset{cmd\ disp\ gap}{\twoheadrightarrow} \{\langle\tau_{s_2}, 0.07, cmd\ disp\ gap(cmd)\rangle,$$
$$\langle\tau_{s_1}, 0.9021, cmd\ disp\ gap(succ)\rangle, \langle\tau_{s_1}, 0.0279, cmd\ disp\ gap(fail)\rangle\}$$

because

$$\{\langle\tau_{s_1}, 1, \sigma\rangle\} \overset{cmd}{\twoheadrightarrow} \{\langle\tau_{s_2}, 1, cmd\rangle\} \overset{disp}{\twoheadrightarrow} \{\langle\tau_{s_1}, 0.07, cmd\ disp\rangle, \langle\tau_{s_3}, 0.93, cmd\ disp\rangle\} \overset{gap}{\twoheadrightarrow}$$
$$\{\langle\tau_{s_2}, 0.07, cmd\ disp\ gap(cmd)\rangle, \langle\tau_{s_1}, 0.9021, cmd\ disp\ gap(succ)\rangle,$$
$$\langle\tau_{s_1}, 0.0279, cmd\ disp\ gap(fail)\rangle\}$$

4 From HMMs to PTEs

A PTE where probabilities associated with event types are consistent with their intended meaning and with the probability properties might be complex when written from scratch. Besides needing a deep knowledge of the modeled system, the developer would also need a means to ensure that, for example, a PTE like $cmd[0.9] : \tau_{s_1} \vee disp[0.8] : \tau_{s_2}$ is recognized as wrong, since there are two mutually exclusive branches and the sum of their probabilities is greater than one. While this error is trivial and can be easily catched and corrected, if the PTE grows in size and complexity a manual development becomes more and more error-prone.

A good practice in engineering new software applications is to reuse well established approaches as much as possible. Even if we want to model probabilistic systems using an extension of Trace Expressions, which is more expressive than HMM and deterministic finite state machines, this does not prevent us from starting from a less expressive but widely used formalism like HMM in order to create a simple, but correct, PTE modeling the system, and extend/refine the PTE if necessary.

If an HMM representing the behaviour of the modeled system exists, for example because it has been learned using existing algorithms, we can indeed use it to generate the corresponding PTE in an automatic way. Once such PTE has been obtained, we can modify it in order to model those features of the actual system that could not be directly represented with an HMM. Ensuring consistency of the modifications is up to the developer.

The HMM2PTE Algorithm. Given an HMM $H = \langle S, A, V, B, \Pi \rangle$, the algorithm to construct an equivalent PTE is the following:

1. for each observation symbol $v_k \in V$, generate the corresponding singleton event type $\beta_k = \{v_k\}$ (recall that Trace Expressions are defined on top of event types and not of events);
2. for each $i = 1..N_s$, for each $j = 1..N_s$, for each $k = 1..N_v$, if $A_{i,j} \neq 0$ then $\tau_{s_i} = \bigvee_{j=1..N_s, k=1..N_v} \beta_k[A_{i,j} * b_i(v_k)]{:}\tau_{s_j}$[2]. If, for some given i, there exists only one j such that $A_{i,j}$ is different from 0, then $\tau_{s_i} = \beta_k[A_{i,j} * b_{i,k}]{:}\tau_{s_j}$. If, for some given i, all $A_{i,j}$ are equal to 0, then $\tau_{s_i} = \epsilon$.

As an example, the HMM2PTE algorithm translates the HMM presented in Sect. 2 into the PTE τ'_{init} presented in Sect. 3.

Forward Algorithm for Probabilistic Trace Expressions. Let us consider the set of PTEs states $\Gamma_0 = \{\langle \tau_{s_1}, \pi_{s_1}, \sigma \rangle, \langle \tau_{s_2}, \pi_{s_2}, \sigma \rangle, ..., \langle \tau_{s_N}, \pi_{s_N}, \sigma \rangle\}$, where each τ_{s_i} corresponds to a state s_i in the HMM H and has been obtained applying the HMM2PTE translation algorithm to H. π_{s_i} is the initial probability of s_i, according to H. If $\pi_{s_i} = 0$, the corresponding state $\langle \tau_{s_i}, \pi_{s_i}, \sigma \rangle$ is not included in Γ_0.

If $\Gamma_0 \overset{O_1...O_{t-1}}{\twoheadrightarrow} \Gamma_{t-1}$, all the states in Γ_{t-1} must have the form $\langle \tau_{s_x}, \pi, O_1...O_{t-1} \rangle$ for some x: they are the states where τ_{s_x} can be reached from one of the states in Γ_0, upon observing $O_1...O_{t-1}$. Given i_1 and i_2 two indexes, we denote with $\Gamma_{i_1}(\tau_{i_2}) = \{\langle \tau_{s_{i_2}}, \pi, O_1...O_{i_1-1} \rangle \in \Gamma_{i_1}\}$.

Theorem 1. *If* $\Gamma_0 \overset{O_1...O_{t-1}}{\twoheadrightarrow} \Gamma_{t-1}$ *and* $\Gamma_{t-1}(\tau_{s_t}) \overset{O_t}{\twoheadrightarrow} \Gamma_t$, *then* $\alpha_t(j) = \Sigma_{\langle \tau_{s_j}, \pi_j, O_1...O_t \rangle \in \Gamma_t} \pi_j$.

We give the intuition behind the theorem by means of our running example. Let us suppose that we want to compute the probability that, after observing

[2] By $\bigvee_{h=1..m} \tau_h$ we mean the conjunction via the $\vee$ operator of the Trace Expressions $\tau_1, ..., \tau_m$. The notation can only be used if $m \geq 2$.

`command(a, start, 0)` (C in the sequel), `dispatch(a, start, 1)` (D in the sequel), `fail(a, start, 2)` (F in the sequel), the system is in state s_3.

Step 1: computation of $\Gamma_0 \overset{O_1 \ldots O_{t-1}}{\twoheadrightarrow} \Gamma_{t-1}$.
In our example, the first step amounts to computing $\Gamma_0 \overset{CD}{\twoheadrightarrow} \Gamma_2$.

$$\Gamma_0 = \{\langle \tau_{s_1}, 1, \sigma \rangle\} \overset{C}{\twoheadrightarrow} \Gamma_1 = \{\langle \tau_{s_2}, 1, C \rangle\} \overset{D}{\twoheadrightarrow} \Gamma_2 = \{\langle \tau_{s_1}, 0.07, CD \rangle, \langle \tau_{s_3}, 0.93, CD \rangle\}$$

Step 2: computation of $\Gamma_{t-1}(\tau_{s_t}) \overset{O_t}{\twoheadrightarrow} \Gamma_t$.
In our example, this step amounts to computing $\Gamma_2(\tau_{s_3}) \overset{F}{\twoheadrightarrow} \Gamma_3$.
Once reached $\Gamma_2 = \{\langle \tau_{s_1}, 0.07, CD \rangle, \langle \tau_{s_3}, 0.93, CD \rangle\}$ we have to limit the last transition, tagged with F, to those states whose Trace Expression corresponds to s_3, namely τ_{s_3}. We have

$$\Gamma_2(\tau_{s_3}) = \{\langle \tau_{s_3}, 0.93, CD \rangle\} \overset{F}{\twoheadrightarrow} \Gamma_3 = \{\langle \tau_{s_1}, 0.0279, CDF \rangle\}$$

Step 3: computation of $\Sigma_{\langle \tau_{s_j}, \pi_j, O_1 \ldots O_t \rangle \in \Gamma_t} \pi_j$
In the last step, we have to sum all the probabilities of the states in Γ_t, namely Γ_3 in our example. There is only one state in Γ_3, with probability 0.0279. It turns out that $\Sigma_{\langle \tau_{s_j}, \pi_j, O_1 \ldots O_t \rangle \in \Gamma_3} \pi_j = \pi_3 = 0.0279$.

Step 4: computation of $\alpha_t(j)$ as defined in the forward algorithm [29] and summarized in Sect. 2.
In our example, $\alpha_t(j)$ is $\alpha_3(3)$, namely the probability to observe CDF, with F observed in state s_3. We use the sequence of events as subscript for α instead of their indexes for sake of clarity.
The base case leads to the following computation:

$$\alpha_C(1) = \pi_1 * b_1(C) = 1 * 1 = 1$$
$$\alpha_C(2) = \pi_2 * b_2(C) = 0 * 0 = 0$$
$$\alpha_C(3) = \pi_3 * b_3(C) = 0 * 0 = 0$$

The first recursive step leads to the following computation (we omit some details and keep the result)

$$\alpha_{CD}(1) = (\Sigma_{i=1..N_s} \alpha_C(i) A_{i,1}) b_1(D) = 0$$
$$\alpha_{CD}(2) = (\Sigma_{i=1..N_s} \alpha_C(i) A_{i,2}) b_2(D) = 1 * A_{1,2} * b_2(D) = 1 * 1 * 1 = 1$$
$$\alpha_{CD}(3) = (\Sigma_{i=1..N_s} \alpha_C(i) A_{i,3}) b_3(D) = 0$$

and the second recursive step leads to

$$\alpha_{CDF}(1) = (\Sigma_{i=1..N_s} \alpha_{CD}(i) A_{i,1}) b_1(F) = 0$$

$$\alpha_{CDF}(2) = (\Sigma_{i=1..N_s}\alpha_{CD}(i)A_{i,2})b_2(F) = 0$$
$$\alpha_{CDF}(3) = (\Sigma_{i=1..N_s}\alpha_{CD}(i)A_{i,3})b_3(F) = \alpha_{CD}(2)*A_{2,3}*b_3(F) = 1*0.97*0.03 = 0.0279$$

Step 5: check that $\alpha_t(j)$ and $\Sigma_{\langle\tau_{s_j},\pi_j,O_1...O_t\rangle\in\Gamma_t}\pi_j$ are equal.
From Steps 3 and 4 above, we obtain $\Sigma_{\langle\tau_{s_3},\pi_3,CDF\rangle\in\Gamma_{CDF}}\pi_3 = 0.0279$ and $\alpha_{CDF}(3) = 0.0279$: for this example the theorem is satisfied.

Proof: the proof of Theorem 1 is reported in Appendix B of the extended version of this paper [8].

Satisfying LTL Properties when Gaps Are Observed. In order to verify whether a LTL property ϕ is verified by a PTE τ, also in presence of observation gaps, we need to specify ϕ into the same formalism in which τ has been modelled, namely PTEs.

The pipeline for implementing the translation from ϕ into an equivalent PTE $\tau(\phi)$ is the following:

1. translate ϕ into a non probabilistic Trace Expression $\tau_{np}(\phi)$ using the implemented algorithm presented in [5];
2. translate the non probabilistic Trace Expression $\tau_{np}(\phi)$ into a probabilistic Trace Expression $\tau(\phi)$ using the "Probabilize" implemented algorithm presented below.

The first step above returns by construction a Trace Expression $\tau_{np}(\phi)$ modeled as a set of equations $\tau_{np}(\phi)_1, ..., \tau_{np}(\phi)_K$, where $\tau_{np}(\phi) = \tau_{np}(\phi)_1$ and each $\tau_{np}(\phi)_i$ has the following form: $\tau_{np}(\phi)_i = \vartheta_{i1}{:}X_{i1} \vee \vartheta_{i2}{:}X_{i2} \vee ... \vee \vartheta_{iK}{:}X_{iK}$.

X_{iK} can in turn be one of the $\tau_{np}(\phi)$ variables, or the constant Trace Expression 1 defined in Sect. 2.

Probabilize correctly terminates on Trace Expressions of this form. If run on Trace Expressions which contain "$\wedge$", "$|$" and "$\cdot$" operators, or that just do not meet the structure above, *Probabilize* fails.

Given a non probabilistic Trace Expression $\tau_{np}(\phi)$, we can obtain its corresponding probabilistic version $\mathit{Probabilize}(\tau_{np}(\phi))$ by adding probability parameters to all the event types that appear in the disjuncts of $\tau_{np}(\phi)$. To achieve this result, we have to define an algorithm that operates on τ_{np} following its structure and that, when there are more than one possible moves from the current state to the next ones due to observability of different event types, shares the probability among these event types following some probability distribution, the uniform one in the simplest case. For instance, if the algorithm is currently analyzing the state $\mathit{cmd} : \tau_{s_1} \vee \mathit{disp} : \tau_{s_2}$ and if it is using a uniform distribution probability,

$$\mathit{Probabilize}(\mathit{cmd} : \tau_{s_1} \vee \mathit{disp} : \tau_{s_2}) = \mathit{cmd}[0.5] : \tau_{s_1} \vee \mathit{disp}[0.5] : \tau_{s_2}$$

If uniform distribution probability is adopted, the structure-driven definition of *Probabilize* is the following:

$$\mathit{Probabilize}(\vartheta_{i1}{:}X_{i1} \vee \vartheta_{i2}{:}X_{i2} \vee ... \vee \vartheta_{iK}{:}X_{iK}) =$$

$$\vartheta_{i1}[1/K]{:}Pr(X_{i1}) \lor \vartheta_{i2}[1/K]{:}Pr(X_{i2}) \lor \ldots \lor \vartheta_{iK}[1/K]{:}Pr(X_{iK})$$

where $Pr(X_{ij}) = X_{ij}$ if $X_{ij} \neq 1$, and $Pr(X_{ij}) = \epsilon \lor everyEvent[1]{:}X_{ij}$ otherwhise. Because of the special form of $\tau_{np}(\phi)$, and the absence of operators besides ":" and "$\lor$" therein, the simple rule above is the only one we need for defining *Probabilize*.

Given these ingredients, satisfaction of LTL properties in presence of observation gaps can be verified in a natural and straightforward way thanks to

- the possibility to represent a LTL property as a standard Trace Expression,
- the possibility to transform such a Trace Expression into a probabilistic one thanks to the *Probabilize* algorithm, and
- the "and" operator, $\land$, modeling the fact that the (probabilistic) Trace Expressions in the two branches must perform the same transitions. From an set-theoretic viewpoint, $\land$ models the intersection of the event traces represented by the two branches it joins.

Let us identify with τ_{HMM} the PTE representing an HMM, and with $\tau_{np}(\phi)$ the standard Trace Expression representing the temporal property ϕ to be verified.

The PTE $\tau_{HMM} \land Probabilize(\tau_{np}(\phi))$ models the intersection of traces of events consistent with the HMM and traces of events that satisfy ϕ: by making the intersection of the states in τ_{HMM} with those in $Probabilize(\tau_{np}(\phi))$ we automatically constrain the evaluation process to those traces produced by the HMM that respect ϕ.

5 Minding Gaps in a Centralized Setting

All the algorithms presented in the previous sections have been implemented using SWI-Prolog[3]. The code and the examples used for our experiments can be downloaded from https://vivianamascardi.github.io/Software/PTE.pl.

PTEs can be modelled as Prolog terms; by exploiting syntactic equations where the same variable appears both to the left and to the right of the "=" syntactic equality symbol, recursive PTEs can be easily defined. This feature is supported by most Prolog implementations, including SWI-Prolog, and allows us to define the PTEs shown in the examples provided so far, with almost the same syntax used in the paper. The adoption of Prolog is a winning choice not only for representing PTEs, but also for implementing their semantics and for manipulating them. Thanks to Prolog's rule-based, declarative interpretation, the rules defining PTE operational semantics have a one-to-one correspondence with Prolog clauses: backtracking and "all-solutions" predicates are powerful tools to deal with the generation of multiple PTE states, when gaps introduce nondeterminism (*set-to-set* rule). A SWI-Prolog PTE-driven monitor observing events taking place in the system under verification, and checking whether they

[3] http://swi-prolog.org/.

comply with the PTE or not, can be automatically generated from the PTE Prolog representation. Connectors with such SWI-Prolog PTE-driven monitors exist both for MASs [2,20,21] and for other systems, including Internet of Things [9,26] and object oriented applications [3]. So far, the algorithms for RV of partially observable MASs have been tested in a simulated environment, namely, with no real connection with implemented systems.

Events can be observed as an online stream while they are generated by the system (*online RV*), or can be recorded on a log file and then inspected (*offline RV*). In both scenarios there may be gaps, due to different reasons. In offline RV, gaps might be caused by event sampling, as usually done to reduce the monitor workload. In online RV, a gap indicates lack of information (a lost message, event or perception); in this case, the absence of information may be due to technical constraints of the system or of the monitor observation capabilities rather than to optimization purposes.

The *set-to-set* semantic rule generates a set of states each time it is applied. The states are maintained by SWI-Prolog in its local knowledge base, to allow the monitor to retrieve the current set of states, query each of them, and update the knowledge base with newly generated states. Unfortunately, a rule like *set-to-set* suffers from state space explosion, in particular when there are many sources of nondeterminism. Each time a gap takes place, the monitor must make guesses on the possible actual events that the gap represents and save all the states generated by these guesses. A possibly huge logical tree-like structure with states as nodes, and moves from states to states as edges, represents these open possibilities. If RV takes place online, the exploration of this logical structure must follow a breadth-first strategy (more space needed but possibly less time required to recognize that the trace is not compliant with the expected behaviour), as the final trace of events is unknown and the levels of the structure are generated and explored at the same time. When, instead, a log file is analyzed offline, the trace in the log is already complete and the logical tree-like structure can be explored, looking for violations, following a depth-first search (less space needed, but the violation could be discovered after exploring all the structure).

Online RV is definitely more challenging: if the log file is analyzed offline, after the system has completed its execution, discovering a violation with some (further) delay is not an issue. But if RV takes place online, it must be performed as efficiently as possible, and in such a way that violations are discovered as soon as possible, to take actions including repairing the system if possible or even stopping its execution, to avoid more serious consequences. This paves the way to two more scenarios: centralized online RV, discusses in this paper, and decentralized online RV, discussed in the companion paper presented at CILC 2022 [7]. In this section we present the reader with an example to understand how a centralized PTE-driven monitor works, and what the state explosion problem means in practice.

Let us consider a MAS involving four agents: $\{alice, bob, charlie, dave\}$. The set of events of our interest in this scenario is the set of messages $Msgs$ that these agents can use to communicate with each other. Such events can be represented as $a_1 \overset{c}{\Longrightarrow} a_2$, meaning that agent a_1 sends a message to a_2 with content c. Since messages are composed by (at least) three mandatory components, sender, receiver and content, besides the totally uninstantiated gap where nothing is known, there may be many partially instantiated gaps such as:

- $gap(a_1 \overset{\_}{\Longrightarrow} a_2)$, where the content of the message is unknown;
- $gap(\_ \overset{m}{\Longrightarrow} a_2)$, where the sender is unknown;
- $gap(a_1 \overset{m}{\Longrightarrow} \_)$, where the receiver is unknown.

In order to make the presentation easier to read we consider event types containing only one message (singleton): instead of writing for example $\vartheta{:}\tau$ where $[\![\vartheta]\!] = \{alice \overset{m_1}{\Longrightarrow} bob\}$ (event type representing the message from *alice* to *bob* with content m_1), we directly write $alice \overset{m_1}{\Longrightarrow} bob{:}\tau$. Given the PTE

$$\tau = \tau_1 \vee \tau_2$$

$$\tau_1 = alice \overset{msg_1}{\Longrightarrow} bob[0.7] : (bob \overset{msg_2}{\Longrightarrow} charlie[0.6] : \tau_1 \mid bob \overset{msg_3}{\Longrightarrow} dave[0.4] : \epsilon)$$

$$\tau_2 = alice \overset{msg_4}{\Longrightarrow} dave[0.3] : (charlie \overset{msg_5}{\Longrightarrow} dave[0.3] : \epsilon \mid bob \overset{msg_3}{\Longrightarrow} dave[0.7] : \tau_2)$$

and initial probability of τ equal to 1, a centralized monitor M_c observing all the interactions among the agents starting from the state τ would behave in the following way. Let us identify with $M_{0,c}$ where c stands for "centralized", the initial state of M_c. $M_{0,c} = \langle \tau, 1, \sigma \rangle$. We highlight that τ contains the shuffle operator $|$ and hence cannot be the output of the HMM2PTE algorithm. It has been designed "by hand", to show that PTEs can be also designed and developed from scratch, besides being automatically generated from a HMM. Being very simple, we can easily check that it is consistent w.r.t. the properties that probability of events must ensure. In the general case, a manual consistency check may be hard to carry out, and its automation is out of the scope of this paper. Let us suppose that the first observed event is a totally uninstantiated gap. Starting from τ, the only two possible evolutions of the protocol are those where either *alice* sends msg_1 to *bob* ($alice \overset{msg_1}{\Longrightarrow} bob$) or *alice* sends msg_4 to *dave* ($alice \overset{msg_4}{\Longrightarrow} dave$). These evolutions may be formalized as (we use *ch* instead of *charlie* for space constraints)

$$M_{0,c} \overset{gap}{\twoheadrightarrow} M_{1,c} = \{\langle bob \overset{msg_2}{\Longrightarrow} ch[0.6] : \tau_1 \mid bob \overset{msg_3}{\Longrightarrow} dave[0.4] : \epsilon, 0.7, gap(alice \overset{msg_1}{\Longrightarrow} bob)\rangle,$$
$$\langle ch \overset{msg_5}{\Longrightarrow} dave[0.3] : \epsilon \mid bob \overset{msg_3}{\Longrightarrow} dave[0.7] : \tau_2, 0.3, gap(alice \overset{msg_4}{\Longrightarrow} dave)\rangle\}$$

If another totally uninstantiated gap is observed, each state in $M_{1,c}$ can evolve in two different ways because of the shuffle, leading to

$$M_{1,c} \overset{gap}{\twoheadrightarrow} M_{2,c} = \{\langle \tau_1 \mid bob \overset{msg_3}{\Longrightarrow} dave[0.4] : \epsilon, 0.42, gap(alice \overset{msg_1}{\Longrightarrow} bob)\ gap(bob \overset{msg_2}{\Longrightarrow} ch)\rangle,$$
$$\langle bob \overset{msg_2}{\Longrightarrow} ch[0.6] : \tau_1, 0.28, gap(alice \overset{msg_1}{\Longrightarrow} bob)\ gap(bob \overset{msg_3}{\Longrightarrow} dave)\rangle,$$

$$\langle bob \overset{msg3}{\Longrightarrow} dave[0.7] : \tau_2, 0.09, gap(alice \overset{msg4}{\Longrightarrow} dave)\ gap(ch \overset{msg5}{\Longrightarrow} dave)\rangle$$
$$\langle ch \overset{msg5}{\Longrightarrow} dave[0.3] : \epsilon \mid \tau_2, 0.21, gap(alice \overset{msg4}{\Longrightarrow} dave)\ gap(bob \overset{msg3}{\Longrightarrow} dave)\rangle\}$$

It is easy to see that the number of states can rapidly grow, because one single monitor is in charge for the RV of all the MAS and takes care of all the possibilities that open up when gaps are observed, that is the main limitation and bottleneck of the approach implemented so far. One approach to cope with state space explosion is to split the centralized monitor into a set of decentralized ones, each observing a portion of the MAS. Since each decentralized monitor has to make its guesses about gaps, when a gap is observed there may be different opinions about its possible values. With respect to a centralized approach, different perspectives due to decentralization need to be managed through synchronization between the monitors, which generates some communication overhead. The algorithm for "minding gaps in a decentralized way" is presented in [7]. The experiments presented in that paper show that, despite the communication overhead due to synchronization, decentralization reduces the search space, in particular when the number of components that generate observable events in the system, be them agents, actors, artefacts, sensors, increases.

6 Conclusions and Future Work

In this paper, we addressed the presence of gaps in observed traces and the need to estimate the probability that the (incomplete) traces satisfy some LTL properties, when the system is modelled by a PTE.

Differently from the work by Stoller at al. we took inspiration from [32], to perform runtime verification using PTEs, we need that each gap represents one single unobserved event: if we have a sequence of three unobserved events, we must have three different gaps in the observed trace. If, in the real system, this one event-one gap correspondence cannot be achieved, we should estimate the number of unobserved events that took place in a time slot T by computing the average rate of the event generation G, and inserting $T * G$ gaps in the event trace. As an example, if the monitor pauses for 3 s and the average events generation rate is 4 events for second, the trace should have 12 consecutive gaps corresponding to what happened in the time slot T.

Although PTEs have a higher potential expressive power than HMM and LTL, being able to express traces like $a^n b^n c^n$, in this work we start from an HMM of the real system and generate an equivalent PTE from it, which of course is as expressive as the HMM it originates from. This is a safe approach to generate a PTE consistent with the known probability distribution of events, which can then be refined in such a way that its expressivennes is fully exploited. Providing guidelines and automatic tools to support the developer in this refinement step is part of our future investigations: we plan to extend RIVERtools [4] towards this direction. More urgent, both the centralized and the decentralized versions of the algorithm have been experimented in a simulated setting; implementing them on top of a real MAS framework like JADE [17] or Jason [19] is the first item in our agenda.

References

1. Ancona, D., Briola, D., Ferrando, A., Mascardi, V.: Global protocols as first class entities for self-adaptive agents. In: AAMAS, pp. 1019–1029. ACM (2015)
2. Ancona, D., Drossopoulou, S., Mascardi, V.: Automatic generation of self-monitoring MASs from multiparty global session types in Jason. In: Declarative Agent Languages and Technologies X - 10th International Workshop, DALT 2012, Valencia, Spain, 4 June 2012, Revised Selected Papers, pp. 76–95 (2012)
3. Ancona, D., Ferrando, A., Franceschini, L., Mascardi, V.: Parametric trace expressions for runtime verification of Java-like programs. In: FTfJP@ECOOP, pp. 10:1–10:6. ACM (2017)
4. Ancona, D., Ferrando, A., Franceschini, L., Mascardi, V.: Managing bad AIPs with RIVERtools. In: Demazeau, Y., An, B., Bajo, J., Fernández-Caballero, A. (eds.) PAAMS 2018. LNCS (LNAI), vol. 10978, pp. 296–300. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94580-4_24
5. Ancona, D., Ferrando, A., Mascardi, V.: Comparing trace expressions and linear temporal logic for runtime verification. In: Ábrahám, E., Bonsangue, M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods. LNCS, vol. 9660, pp. 47–64. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30734-3_6
6. Ancona, D., Ferrando, A., Mascardi, V.: Parametric runtime verification of multi-agent systems. In: AAMAS, pp. 1457–1459. ACM (2017)
7. Ancona, D., Ferrando, A., Mascardi, V.: Exploiting probabilistic trace expressions for decentralized runtime verification with gaps. In: The 37th Italian Conference on Computational Logic, CILC 2022, CEUR Workshop Proceedings. CEUR-WS.org (2022)
8. Ancona, D., Ferrando, A., Mascardi, V.: Mind the gap! Runtime verification of partially observable MASs with probabilistic trace expressions - extended version. Technical report, University of Genova, DIBRIS (2022). This technical report extends the contents of this EUMAS 2022 paper with two appendices. https://vivianamascardi.github.io/Documents/technical-report-EUMAS2022.pdf
9. Ancona, D., Franceschini, L., Delzanno, G., Leotta, M., Ribaudo, M., Ricca, F.: Towards runtime monitoring of Node.js and its application to the Internet of Things. In: ALP4IoT@iFM. EPTCS, vol. 264, pp. 27–42 (2017)
10. Ancona, D., Franceschini, L., Ferrando, A., Mascardi, V.: RML: theory and practice of a domain specific language for runtime verification. Sci. Comput. Program. **205**, 102610 (2021)
11. Babaee, R., Gurfinkel, A., Fischmeister, S.: *Prevent*: a predictive run-time verification framework using statistical learning. In: Johnsen, E.B., Schaefer, I. (eds.) SEFM 2018. LNCS, vol. 10886, pp. 205–220. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92970-5_13
12. Barringer, H., Groce, A., Havelund, K., Smith, M.H.: Formal analysis of log files. JACIC **7**(11), 365–390 (2010)
13. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification - Introductory and Advanced Topics. LNCS, vol. 10457. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5
14. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 1–33. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_1

15. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 326–340. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_30
16. Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state Markov chains. Ann. Math. Statist. **37**(6), 1554–1563 (1966)
17. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley, Hoboken (2007)
18. Bonakdarpour, B., Fraigniaud, P., Rajsbaum, S., Travers, C. (eds.) Bertinoro Seminar on Distributed Runtime Verification, May 2016 (2016). http://www.labri.fr/perso/travers/DRV2016/
19. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley, Hoboken (2007)
20. Briola, D., Mascardi, V., Ancona, D.: Distributed runtime verification of JADE and Jason multiagent systems with Prolog. In: CILC. CEUR Workshop Proceedings, vol. 1195, pp. 319–323. CEUR-WS.org (2014)
21. Briola, D., Mascardi, V., Ancona, D.: Distributed runtime verification of JADE multiagent systems. In: Camacho, D., Braubach, L., Venticinque, S., Badica, C. (eds.) Intelligent Distributed Computing VIII. SCI, vol. 570, pp. 81–91. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10422-5_10
22. Cairoli, F., Bortolussi, L., Paoletti, N.: Neural predictive monitoring under partial observability. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 121–141. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88494-9_7
23. Havelund, K., Leucker, M., Sachenbacher, M., Sokolsky, O., Williams, B.C. (eds.) Runtime Verification, Diagnosis, Planning and Control for Autonomous Systems, 07.11. - 12.11.2010. Dagstuhl Seminar Proceedings, vol. 10451. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2010)
24. Havelund, K., Reger, G., Rosu, G.: Runtime verification - past experiences and future projections. In: Special Issue in Celebration of Issue Number 10,000 of Lecture Notes in Computer Science. LNCS, vol. 10000 (2018)
25. Joshi, Y., Tchamgoue, G.M., Fischmeister, S.: Runtime verification of LTL on lossy traces. In: SAC, pp. 1379–1386. ACM (2017)
26. Leotta, M., Ancona, D., Franceschini, L., Olianas, D., Ribaudo, M., Ricca, F.: Towards a runtime verification approach for internet of things systems. In: Pautasso, C., Sánchez-Figueroa, F., Systä, K., Murillo Rodríguez, J.M. (eds.) ICWE 2018. LNCS, vol. 11153, pp. 83–96. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03056-8_8
27. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. **78**(5), 293–303 (2009)
28. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October–1 November 1977, pp. 46–57. IEEE Computer Society (1977)
29. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. In: Waibel, A., Lee, K.-F. (eds.) Readings in Speech Recognition, pp. 267–296. Morgan Kaufmann Publishers Inc., San Francisco (1990)
30. Rabiner, L.R., Juang, B.-H.: An introduction to hidden Markov models. IEEE ASSP Mag. 4–16 (1986)
31. Sánchez, C., et al.: A survey of challenges for runtime verification from advanced application domains (beyond software). Formal Methods Syst. Des. **54**(3), 279–335 (2019)

32. Stoller, S.D., et al.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_15