Automatic Compositional Verification of Probabilistic Safety Properties for Inter-organisational Workflow Processes

Redouane Bouchekir[1], Saida Boukhedouma[2] and Mohand Cherif Boukala[1]

[1]*MOVEP, Computer Science department, USTHB, BP 32 El-Alia, Algiers, Algeria*

[2]*LSI, Computer Science department, USTHB, BP 32 El-Alia, Algiers, Algeria*

{rbouchekir,sboukhedouma,mboukala}@usthb.dz

Keywords: Verification, IOWF process, BPEL4WS, Pobabilistic automata, model checking, compositional verification, assume-guarantee reasoning rule.

Abstract: For many complex systems, it is important to verify formally their correctness; the aim is to guarantee the reliability and the correctness of such systems before their effective deployment. Several methods have been proposed to this effect using different formal tools such as Probabilistic Automata (PA). In this paper we focus on verification of service-based inter-organizational workflow (IOWF) processes which support collaboration and cooperation between WF processes attached to several partners, and specified using the business process execution language (BPEL4WS). Then, IOWF processes are translated to Probabilistic Automata (PA) models. More than verification support, PA provides a numerical evaluation of the IOWF process. We also propose the use of compositional verification to cope with the state space explosion problem. The IOWF behavior is checked against probabilistic safety properties. The verification and the analysis are performed in an automated way using the PRISM model checker and based on the assume-guarantee reasoning rule.

1 INTRODUCTION

A general characteristic of modern computing systems is their complexity where chances of introducing errors increase significantly. Detecting and fixing these errors is a very important goal. *Model checking* is an effective method to verify systems by exhaustively searching the complete state space exhibited by a system. A variant of model checking is *probabilistic model checking* (Baier and Katoen, 2008), (Baier and Kwiatkowska, 1998).

Probabilistic model checking involves the construction of a finite-state model augmented with probabilistic information, such as Markov chain or PA (Lehmann and Shelah, 1982), (Hart et al., 1984). This model is then checked against properties specified in probabilistic extensions of temporal logic, such as PCTL (Hansson and Jonsson, 1994). This permits quantitative notions of correctness to be checked, e.g. " The message will be delivered with probability 1". It also provides other classes of properties, such as performance or reliability, e.g. "compute expected probability for a successful transmission of a data packet".

As with any formal verification technique, one of the main challenge for probabilistic model checking when checking large/complex systems is *the state space explosion problem*: the number of states grows exponentially. A promising direction to combat this problem is the *compositional verification* (Larsen et al., 1995), (Feng et al., 2010), (Feng, 2013).

Compositional verification suggests a "divide and conquer" strategy to reduce the verification task into simpler subtasks. A popular approach is the *assume-guarantee* paradigm (AG) (Păsăreanu et al., 2008), (Chen et al., 2010), in which individual system components are verified under *assumptions* about their environment. Once it has been verified that all system components do indeed satisfy these assumptions, proof rules can be used to combine individual verification results, establishing correctness properties of the overall system.

For systems modelled using PA, AG reasoning proof rules are recently proposed in (He et al., 2015), (Feng, 2013), (Feng et al., 2010).

Inter-Organisational Workflow (IOWF) processes constitute a good example of complex systems in the area of B2B cooperation. Indeed, with the emergence of service oriented architecture (SOA) and Web services standards (Alonso et al., 2004), many research works have been directed towards WF composition, mainly based on Web services (Sheng et al., 2014), (Gabrel et al., 2013), (Gorton et al., 2009). Each partner implements his local WF process as an orchestration of Web services. Then, an IOWF process can be considered as a composition of orchestrations of Web services, usually specified using BPEL4WS language (Business Process Execution Language for Web Services) (Jordan et al., 2007). This results in *complex* WF process models. However, there is no guarantee that the specification of a composite WF is free of errors, even if all WF components are safe. Therefore, analysing and verifying the correctness of the IOWF process behaviour before it becomes operational is important.

The work described in this paper focuses on the formal modelling and verification of composite WF (IOWF) processes by using an automatic AG reasoning approach. We need to analyse and verify the composition because each partner aims to be autonomous and have the freedom to create or modify his own WF process at any moment. In addition, this composition should not influence the current WF performance. More precisely, our main goal is to verify the logic correctness of a WF composition, e.g. avoidance unsafe system states, satisfaction of certain business constraints, e.g. a partner needs to receive always a reply after a request. In addition, We use the AG reasoning approach to cope with the the state space explosion problem.

The remainder of the paper is organized as follows: Section 2 summarizes briefly some related works and explains the motivation of this paper. In Section 3, we provide some background on WF, IOWF, PA, the L^* learning algorithm and probabilistic model checking. Section 4 presents our approach to modelling and verifying the IOWF behaviour and finally, Section 5 concludes the paper and talks about future works.

2 RELATED WORKS AND MOTIVATION

Research on formal verification has attracted considerable attention. In this section, we review some research works related to formal verification of WF composition, focusing on model checking and AG reasoning approaches. Authors of (Sbaï et al., 2010) present a method for verifying collaborative WF processes based on model checking techniques. In particular, they propose to verify soundness properties of these processes using SPIN model checker. First, they translate the adopted specification of WFs (i.e. the WF-net) to Promela, which is the description language of models to be verified by SPIN, then express the soundness properties in Linear Temporal Logic (LTL) and use SPIN to test whether each property is satisfied by the Promela model of the WF-net in question. (Doshi et al., 2004) consider the issue of automatic WF composition, they propose to use Markov Decision Processes MDPs to model WF composition. They justify the use of MDPs by : (1) the uncertainty over the true environmental model and, (2) the dynamic environments. Another work (Braghetto et al., 2011) propose to map business process models to stochastic formalisms (Stochastic Automata Network). The main contribution of this work is that they consider both qualitative and quantitative analysis of WF composition; for performance information, they refer to a set of probabilities values attached to the states. In (Gallotti et al., 2008), the authors used a model driven approach, which automatically transforms a design model of service composition into an analysis model, then feeds a probabilistic model checker for quality prediction. The issue of verifying if composite Web services design satisfies some desirable properties was considered also in (Bentahar et al., 2013), the behaviours of Web services are modeled using automata-based techniques (Kripke model). Authors of this work proposed a verification technique based on the reduction of model checking compositions to model checking LTL and CTL, by verifying the operational behaviour against the properties specified by the control behaviour. For the AG approaches, some works propose the use of compositional verification approach (Feng et al., 2010), (Feng, 2013) to cope with the state space explosion problem. An assume-guarantee reasoning rule for PA was proposed in (Feng et al., 2010), (Feng, 2013), using the L^* automatically generate assumptions. Many works used the compositional verification to verify and analyse complex systems, such as (Calinescu et al., 2012), where they proposed to used AG proof rules to verify complex IT systems.

Unlike previous works, in this paper, we state that a well-defined WF schema can be modelled using PA. This choice is advantage by: (1) PAs support the verification and provide a numerical quantitative analysis of IOWF; (2) more large properties can be expressed, e.g. computing the probability to reach unsafe system states, verifying that a partner always receives a reply after a request, with a certain probability value,...etc. (3) PAs can be applied for modelling and verifying large/complex systems (which is the case of the most real-world IOWF processes) by the use of the compositional verification approach. We model, separately, each WF process using PA; the composition of WF is modelled by a synchronisation between PAs. The separate modelling of WF processes composing the global IOWF process will let us to use the AG approach, where each WF is considered as a component.

3 PRELIMINARIES

We give some background knowledge about WF and IOWF in Section 3.1, Section 3.2 presents the PA, Section 3.3 describes the L^* learning algorithm and Section 3.4 presents some basics on the probabilistic model checking.

3.1 Workflow and inter-organizational workflow

A *workflow* (WF) process is the automation of all or part of a business process in which information flows from one activity to another (respectively, one participant to another) according to a set of predefined rules WFMC [1]. The WF behaviour which is defined, in WFMC, as: "*WF schema* is used to represent the structure of an application in terms of tasks as well as temporal and data dependencies between tasks". In practice, a WF schema can be described using BPEL4WS which is an XML-based language that allows composition of Web services in a service-oriented architecture (SOA), for interconnection and data-sharing. Programmers use BPEL4WS for specification and execution of a WF process based on Web services. BPEL4WS messages are typically used to invoke remote services, orchestrate process execution and manage events and exceptions.

Inter-organizational WF (IOWF) can be defined as a manager of activities involving two or more WFs (affiliated with business partners), autonomous, possibly heterogeneous and interoperable in order to achieve a common business goal (Van Der Aalst, 1999).

3.2 Probabilistic automata

In this paper, the PA model is used for modelling WF (and IOWF) schemas. The choice of using PA in our approach comes from the fact that they allow the modelling of the most constructs of IOWF schema, including, basic activities, structured activities and Web service activities. It allows also both the verification and the analysis of IOWF processes. In addition, PAs are known to model concurrent probabilistic systems, which are in our case, IOWF processes allowing concurrent execution of two or more Web services.

Definition 1. *An automata is a tuple $A = (S, s_0, \Sigma_A, \delta_A, F)$ where S is a finite set of states, $s_0 \in S$ is an initial state, Σ_A is a finite set of actions, $\delta_A \subseteq S \times \Sigma_A \times S$ is a transition relation and $F \subseteq S$ is a set of final or accepting states. A is a deterministic finite automata if $|\delta_A(s,a)| \leq 1$ for all $s \in S$ and $a \in \Sigma_A$, where $|\delta_A(s,a)|$ is the size of the set δ_A. A is called a complete DFA if $|\delta_A(s,a)| = 1$.*

In the following, we use $Dist(S)$ to denote the set of all discrete probability distributions over a set S.

Definition 2. *A labelled probabilistic automaton (PA) is a tuple $M = (S, s_0, \Sigma_M, \delta_M, L)$ where S, s_0, Σ_M are defined as for DFA. $\delta_M \subseteq S \times (\Sigma_M \cup \{\tau\}) \times Dist(S)$ is a probabilistic transition relation and $L \rightarrow 2^{AP}$ is a labelling function, assigning atomic propositions form a set AP to a state.*

In a state s of PA M, one or more transitions, denoted $s \underset{\rightarrow}{a} \mu$, are available, where $a \in \Sigma_M$ is an action label, μ is a probability distribution over states and $(s, a, \mu) \in \delta_M$. A path through a PA is a (finite or infinite) sequence $s_0 \underset{\longrightarrow}{a_0, \mu_0} s_1 \underset{\longrightarrow}{a_1, \mu_1} \ldots$ To reason about PAs, we use the notion of adversaries, which resolve the non-deterministic choices in a PA, based on its execution history. Formally, an adversary maps any finite path to a distribution over the available transitions in the last state on the path. In this paper, we use also the notion of parallel composition of PAs, which refers to standard definition of parallel composition of PAs M_1 and M_2, denoted $M_1 \parallel M_2$, where M_1 and M_2 synchronise over all common actions (Segala, 1995).

3.3 The L^* Algorithm

The L^* algorithm (Angluin, 1987) is a formal method to learn a *DFA* with the minimal number of states that accepts an unknown language L over an alphabet Σ. During the learning process, the L^* algorithm interacts with a *Teacher* to make two types of queries: (i) membership queries and (ii) equivalence queries. A *membership queries* are used to check whether some word w is in the target language, we can see membership queries as a function *MQuery* such that:

$$MQuery(w) = \begin{cases} 1 & \text{if } w \in L \\ 0 & \text{otherwise} \end{cases}$$

Equivalence queries are used to check whether a conjectured DFA A accepts the target language. If the conjectured DFA is not correct, the teacher would return a counterexample to L^* to refine the automaton being learnt, also we can see equivalence queries as a function *EQuery* such that:

$$EQuery(A) = \begin{cases} 1 & L(A) = L \\ \text{0 and a counterexample} & \text{otherwise} \end{cases}$$

[1] Workflow Management Coalition-http://www.wfmc.org

At the implementation level, L^* maintains an observation table (U,V). Where U is a non-empty finite prefix-closed of words and V is a non-empty finite suffix-closed set of words. $MQuery(w) = ((U \cup U.\Sigma).V) \mapsto \{+,-\}$ for all $w \in ((U \cup U.\Sigma).V)$, where Σ is the alphabet of the target language and $\{+,-\}$ represent whether or not a word is accepted by the target language or not. The L^* checks at the end of the membership queries whether the observation table is closed and consistent. An observation table T is closed if for all $u \in U$, $v \in V$ and $w \in \Sigma$, there is a $u' \in U$ such that $T(uwv) = T(u'v)$. An observation table T is consistent if for all $u, u' \in U$, $v \in V$ and $w \in \Sigma$, $T(uv) = T(u'v) \Rightarrow T(uwv) = T(u'wv)$.

3.4 Probabilistic model checking

A probabilistic safety properties are specified by Probabilistic Computation Tree Logic PCTL (Hansson and Jonsson, 1994) in the form of $P_{\leq p}[\psi]$ with $p \in [0,1]$ and

$$\phi ::= true|a|\phi_1 \wedge \phi_2|\neg\phi$$
$$\psi ::= \phi U \phi$$

where a is an atomic proposition, ϕ a state formula, ψ a path formula and U the "until" temporal operator.

We consider safety properties, which are often characterized as "nothing bad should happen". In the context of probabilistic systems, safety properties also capture a wide range of useful properties, e.g. "the maximum probability of an error occurring is at most 0.01" is specified as $P_{\leq 0.01}[trueU\phi_{err}]$ where ϕ_{err} is a state formula indicating the occurrence of an error. We say that a state $s \in S$ satisfies P (written $M,s \models P_{\leq p}[trueU\phi_{err}]$) if $P_s^{max} \leq p$. Model M satisfies P (written $M \models P_{\leq p}[trueU\phi_{err}]$) if $M,s_0 \models P_{\leq p}[trueU\phi_{err}]$.

In this paper, we focus on *regular safety property*. Safety property P over AP is called *regular* if its set of bad prefixes constitutes a regular language over 2^{AP}, so a *regular safety property P* can be represented as a set of infinite words, that is characterised by regular language of *bad prefixes*.

4 DESCRIPTION OF OUR APPROACH

In this section, we describe our approach for compositional probabilistic verification of IOWF schema. Figure 1 presents an overview of our approach. The first step is to translate the BPEL4WS files to a PRISM module, where each BPEL4WS file describes a WF process attached to a given partner, by means of PA. The output of the first step is a set of PAs, where each PA models a WF schema. In the second step, we use the AG approach to verify if the IOWF process satisfies the probabilistic safety property P. Indeed, this will let us to check the correctness of the probabilistic safety property P without constructing and analysing the full model, which is $M_{WF_1} \parallel M_{WF_2}$; instead, we generate an assumption A which simulates one component (let A simulates M_{WF_1}). By using the assumption A, if A is much smaller than the component M_{WF_1} then we can expect significant gains in terms of memory space.

4.1 Translate IOWF to PA

In this section, we describe how we translate IOWF schemas to PAs. This translation needs to consider the compositional aspect, this means that we need to model the set of WFs composing the IOWF as a set of components (set of PAs), this will let us to use the AG approach for verification of the global IOWF process. In addition, for automatic translation from BPEL4WS specification to PA model, we have implemented a translator tool. Our translator accepts as input a set of BPEL4WS and WSDL files, and generates a PRISM model, which represents a set of PAs.

The translator translates each WF process to a PRISM module. In the following, we describe how we translate each activity of the WF process to its equivalent in PA and in PRISM code. Also, we describe how we encode communication between WFs invoked in a global IOWF process.

The *initial (final) node* of a WF, is modelled by an *initial (final) state* of the PA.

4.1.1 Modelling basic activities

Basic BPEL4WS control flow activities, which are: *Assign*, *Empty*, *Wait*, *Throw* and *Exit*, are mapped to their equivalents in PAs as described in Table 1. We model the wait activity using flags, since in this paper we aim to model the logic behaviour of an IOWF process. Therefore, we do not need to model time in wait activity.

4.1.2 Modelling structured activities

Usual structured BPEL4WS activities are: *Sequence*, *Switch*, *While*, *Flow*, *Pick*, we mapped each activity to its equivalent in PAs as shown on Table 2.

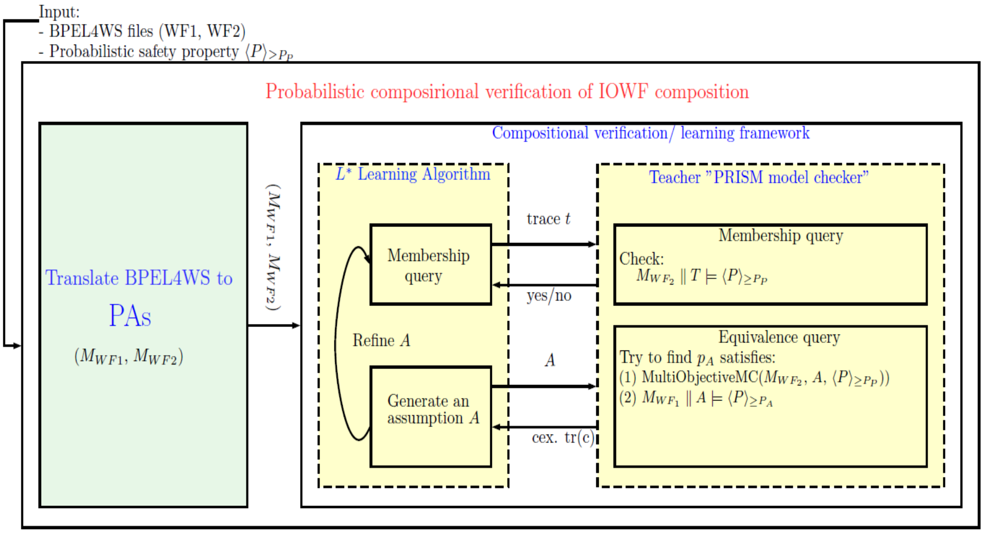
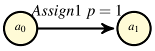
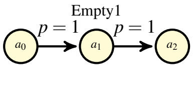
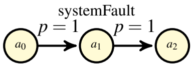

Input:
- BPEL4WS files (WF1, WF2)
- Probabilistic safety property $\langle P\rangle_{>P_P}$
Probabilistic composirional verification of IOWF composition
Translate BPEL4WS to PAs
(M_{WF1}, M_{WF2})
(M_{WF1}, M_{WF2})
Compositional verification/ learning framework
L^* Learning Algorithm
Membership query
Refine A
Generate an assumption A
trace t
yes/no
A
cex. tr(c)
Teacher "PRISM model checker"
Membership query
Check:
$M_{WF_2} \parallel T \models \langle P\rangle_{\geq P_P}$
Equivalence query
Try to find p_A satisfies:
(1) MultiObjectiveMC(M_{WF_2}, A, $\langle P\rangle_{\geq P_P}$))
(2) $M_{WF_1} \parallel A \models \langle P\rangle_{\geq P_A}$

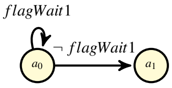
Figure 1: An overview of our approach.

Table 1: Translation of BPEL4WS's basic activities to PRISM language

Name	BPEL4WS Syntax	PAs	PRISM language
Assign	a_0 $< assignname = "Assign1" >$ $< copy >$ $< fromvariable = "Var1"/ >$ $< tovariable = "Var2"/ >$ $< /copy >$ $< /assign >$ a_1		[Assign1] a=0 → (a'=1);

Empty	a_0
	$< emptyname = "Empty1"/ >$
	a_2
			[] a=0 → (a'=1);
			[] a=1 → (a'=2);
			....
			label "Empty1"=a=1;

Throw	a_0
	$< throwname = "Throw1"$
	$faultName = "sxeh : systemFault"$
	$faultVariable = "Variable1"/ >$
	a_2
		 systemFault
		$p = 1$
		$p = 1$
		a_0
		a_1
		a_2
			[] a=0 → (a'=1);
			[] a=1 → (a'=2);
			....
			label "systemFault"=a=1;

Wait	$< waitname = "Wait1" >$
	$< for >' 10' < /for >$
	$< /wait >$
			*formula* flagWait1;
			[] a=0 *and* flagWait1 → (a'=0);
			[] a=0 *and* ¬ flagWait1 → (a'=1);

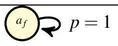
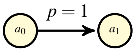
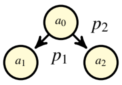
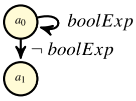
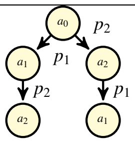
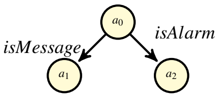
Exit	$< exit/ >$		[final] a=f → (a'=f);

Table 2: Translation of BPEL4WS's structured activities to PRISM language

Activity	BPEL4WS Syntax	PAs	PRISM language
Sequence	$< sequence... >$ a_0 a_1 $< /sequence >$		[] a=0 → (a'=1);
Switch	a_0 $< switch... >$ a_1 a_2 $< /switch >$		[] a=0 → p_1: (a'=1) + p_2: (a'=2); //$p_1 + p_2 = 1$

While	$< whilename = ”While1” >$
	$< condition > boolExp < /condition >$
	a_0
	$< /while >$
	a_1
			*formula* boolExp;
			[] a=0 *and* boolExp → (a'=0);
			[] a=0 *and* ¬ boolExp → (a'=1);

Flow	a_0
	$< flow... >$
	a_1
	a_2
	$< /flow >$
		 a_0
		p_2
		p_1
		a_1
		a_2
		p_2
		p_1
		a_2
		a_1
			[] a=0 → p_1: (a'=1) + p_2: (a'=2); // $p_1 = p_2 = 1$
			[] a=1 → p_2: (a'=2);
			[] a=2 → p_1: (a'=1);

Pick	a_0
	$< pick... >$
	$< onMessage... >$
	a_1
	$< /onMessage... >$
	$< onAlarm... >$
	a_2
	$< /onAlarm... >$
	$< /pick >$
			*formula* isMessage;
			formula isAlarm;
			[] a=0 *and* isMessage → (a'=1);
			[] a=0 *and* isAlarm → (a'=2);

4.1.3 Modelling Web service activities

In BPEL4WS specification, the *”partnerLink”* specifies the partner from/to which a message is received/sent. When the IOWF view is considered, the partnerLink construct plays an important role since it defines the communication channel between the internal/external WF or between WF processes involved in cooperation. Common interactions patterns between WF and external Web services (or external WF) are: *(1) One-way messages, (2) Synchronous interactions* and *(3) Asynchronous interactions*.

One-way messages is considered when a WF process sends a message to an external partner, and this partner does not need to reply. The external partner can be either an external Web service or an external WF (encapsulated in a composite Web service). The WF process sending the message does not wait for a response, but continues executing immediately. Figure 2 shows a one-way interaction. In the case where the external partner is a Web service, we model the one-way messages by a state in the PA, since we do not model the external behaviour of the Web service. Otherwise, we model the one-way interaction between two WF processes by a synchronous interaction between the PAs modelling the two WFs.

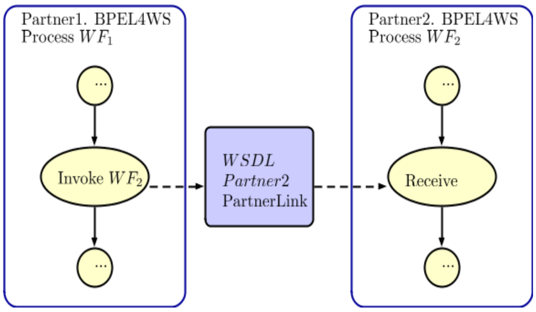

Partner1. BPEL4WS Process WF_1
...
Invoke WF_2
...
$WSDL$ $Partner2$ PartnerLink
Partner2. BPEL4WS Process WF_2
...
Receive
...

Figure 2: One-way interaction.

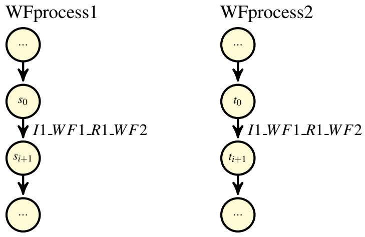


Figure 3: Modelling one-way interaction in PA.

In a synchronous interaction, a WF process sends a request to an external Web service or an external WF by using ”invoke” activity, and receives an immediate reply. Figure 4 shows how we model the synchronous interactions in an IOWF process model.

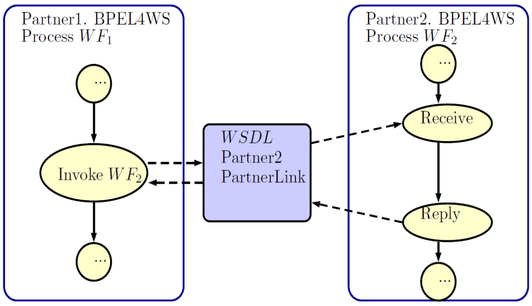

Partner1. BPEL4WS Process WF_1
...
Invoke WF_2
...
WSDL
Partner2
PartnerLink
Partner2. BPEL4WS Process WF_2
...
Receive
Reply
...

Figure 4: Synchronous interaction.

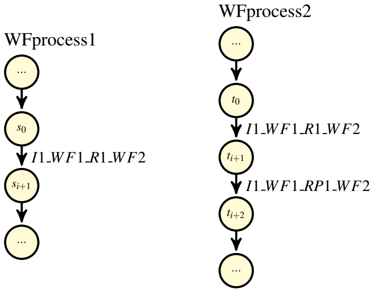


Figure 5: Modelling synchronous interaction in PA.

In an asynchronous interaction, a WF process sends a request to an external Web service or external WF by using the "invoke" activity, and waits until receiving a reply. Figure 6 shows how we model the synchronous interactions in IOWF process model.

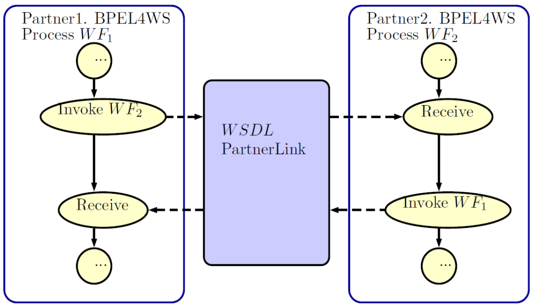


Figure 6: Asynchronous interaction.

4.2 Compositional verification of IOWFs

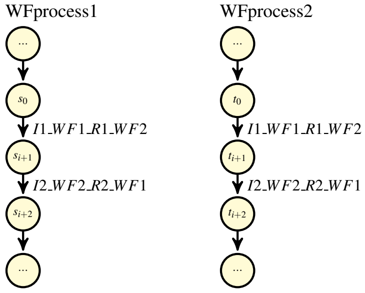


Figure 7: Modelling asynchronous interaction in PA.

The second step in our framework is the compositional verification of IOWF schema. This step is based on an AG reasoning rule, which describes the steps and the rules to infer the assumption A and how to use it in the verification process. For clarity, it is undicidable to infer assumption as PA under a version of Angluin's learning algorithm (L^*), and a learning algorithms for general probabilistic systems may not exist after all (Komuravelli et al., 2012). Given the absence of learning algorithms for probabilistic systems, the authors of (Feng et al., 2010) propose to represent assumption as DFA and use the L^* to infer this assumption. In our work, we adapt the same proof rule to verify the IOWF behaviour.

Theorem 1. *Let M_{WF_1} and M_{WF_2} be PA modelling WF $WF1$ and $WF2$, respectively. $\langle A\rangle_{\geq P_A}$ and $\langle P\rangle_{\geq P_P}$ are regular probabilistic safety properties such that their alphabets satisfy $\Sigma_A \subseteq \Sigma_{M_{WF_1}}$ and $\Sigma_P \subseteq \Sigma_{M_{WF_2}} \cup \Sigma_A, P_p, P_A \in [0,1]$ are a rational probability bound, then the following proof rule holds:*

$$\frac{\begin{array}{lr}\langle true\rangle M_{WF_1}\langle A\rangle_{\geq P_A} & (1)\\ \langle A\rangle_{\geq P_A} M_{WF_1}\langle P\rangle_{\geq P_P} & (2)\end{array}}{\begin{array}{lr}\langle true\rangle M_{WF_1} \parallel M_{WF_2}\langle P\rangle_{\geq P_P} & (3)\end{array}}$$

Table 3: AG rule for probabilistic systems (Feng et al., 2010).

This AG rule means that if we have two PA M_{WF_1} and M_{WF_2} modelling the local WFs WF_1 and WF_2, respectively, composed to build an IOWF process, we can check the correctness of a probabilistic safety property $\langle P\rangle_{\geq P_P}$ on the IOWF without constructing and verifying the full model. Instead, we first infer an appropriate assumption A by using the L^* learning algorithm; then check if we can use this assumption to verify the IOWF by following the two premise (1) and (2). Promise (1) can be checked by using a standard probabilistic model checking algorithm (Kwiatkowska et al., 2009) and promise (2) can be checked using multi-objective model checking algorithm (Etessami et al., 2007).

4.3 Running example

Example 1. *We illustrate the formal modelling and the compositional verification technique dis-*

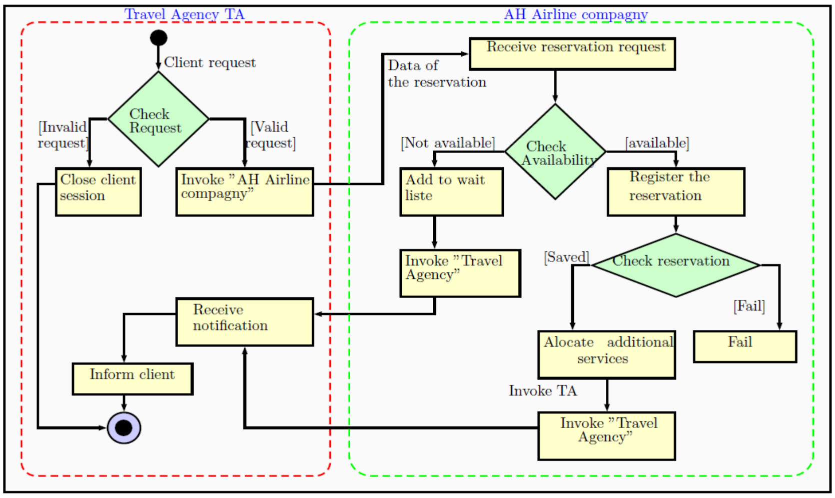


Figure 8: Activity diagram of the inter-organizational workflow schema of *AT/AH*.

cussed in this paper by using the example of inter-organisational process TA/AH shown in Figure 8. It involves two business partners respectively called TA for a Travel Agency and AH for a local airline company. TA provides an internet reservation system for its customers. The TA sends the customer reservation requests to its partner (AH) in case of a valid request; otherwise, it closes the client session. When AH receives a reservation request, it checks if the number of seats requested are available, if they are available, it registers the reservation data and alocate additional services such as seating, special meals ...etc. Otherwise, if the number of seats requested are not available, it adds the request to the waiting list. In both cases, AH company returns a notification to its partner TA.

As described before, the first step in our approach is to model the local WF of TA/AH by using PAs, Figure 9 and 10 shown the PAs modelling the local WF TA and AH, respectively.

For the IOWF AT/AH, we apply the compositional verification to verify if $M_{WF_{TA}} \parallel M_{WF_{AH}}$ satisfies the probabilistic safety property : $P^{min}_{\geq 0.985}[G\neg(\text{"}fail\text{"})]$, which means the minimum probability that the system should never fails, over all possible adversaries, is at least 0.985.

As described before, a probabilistic safety property can be expressed using a regular language of bad prefixes, this can be then represented by a DFA P^{err}, in our case, the DFA specifies our probabilistic safety property is shown in Figure 11.

For instance, we initialize the set $U = V = \{\varepsilon\}$, where U and V are a set of actions used by the L^* learning algorithm to infer a conjecture assumption. $\Sigma_A = \{c, d, e\}$, where c,d and e are the actions

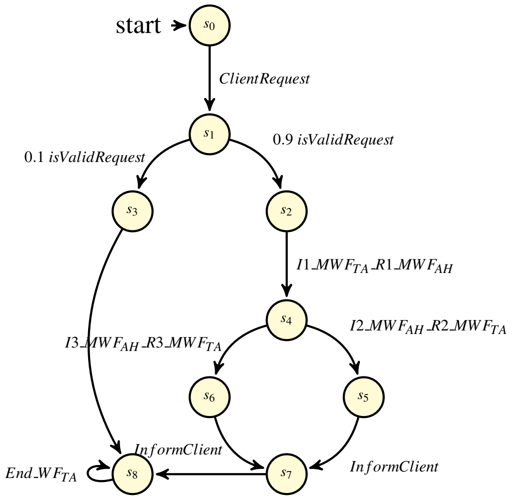


Figure 9: PA $M_{WF_{TA}}$ Modelling the local WF "TA".

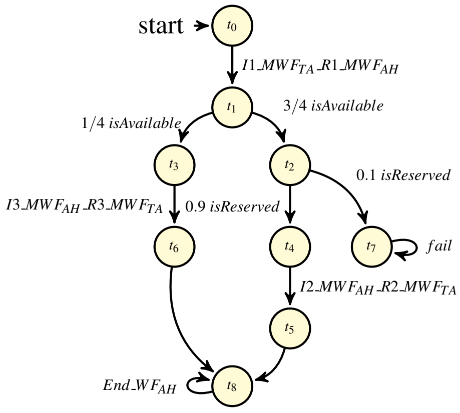


Figure 10: PA $M_{WF_{AH}}$ Modelling the WF "AH".

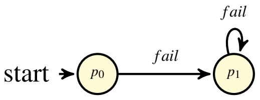


Figure 11: DFA P^{err}.

$I1\_MWF_{TA}\_R1\_MWF_{AH}$, $I2\_MWF_{AH}\_R2\_MWF_{TA}$ and $I3\_MWF_{AH}\_R3\_MWF_{TA}$, respectively. Σ_A represents the set of actions of the assumption and ε is an empty word.

The L^* starts by applying membership queries, which verify whenever a given trace T should be included in the assumption. The trace T denotes a PA representing a trace, i.e a linear, $|T|+1$ state in which each transition has probability 1 and labelled by an action of an element of the set $(U.U).\Sigma_A$. For instance, the element $(U.U).\Sigma_A = \{\varepsilon, c, e, d\}$. L^* starts by checking if c should be included or not in the generated assumption, for this it builds a trace T (shown in Figure 12). The criterion to check if the T should be included in the assumption is checked by verifying if $T \parallel M_{WF_{AH}} \parallel P^{err}$ satisfies the probabilistic safety property P. Consider the trace T shown in Figure 12, we have used the model checker PRISM to check whenever $T \parallel M_{WF_{AH}} \parallel P^{err}$ satisfies the property: $Pmin =?[G\neg((''err_P'')and(''err_T''))]$, where $''err_P''$ and $''err_T''$ are two atomic proposition labelling the final states of P^{err} and T, respectively.

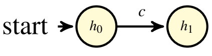


Figure 12: PA representing the trace T for the action c.

The results of the membership queries are stored in an observation table, for the first set of membership queries, the results are stored in the first observation table T_0(Table 4).

T_0	$V = \{\varepsilon\}$
$U = \{\varepsilon\}$	$-$
c	$+$
e	$-$
d	$-$

Table 4: The first observation tables T_0.

The first observation table T_0 is not closed, since there is no row in the table of T_0 that corresponds to $T_0(c)$. To make T_0 closed, we add the action c to the set U. For now, the set U and $(U.U)\Sigma_A$ are set to $U = \{\varepsilon, c\}$ and $(U.U)\Sigma_A = \{\varepsilon, c, e, d, c.c, c.e, c.d\}$. The L^* learning algorithm repeats the same process as before to check whenever $c.c, c.e$ and $c.d$ should be included in the assumption. The second observation Table T_1 is shown in Table 5. The observation table T_1 is closed and consistent.

T_1	$V = \{\varepsilon\}$
$U = \{\varepsilon\}$	$-$
$U = \{c\}$	$+$
e	$-$
d	$-$
$c.c$	$+$
$c.e$	$+$
$c.d$	$+$

Table 5: The second observation tables T_1.

We generate the first conjecture assumption A_1^{err} based on T_1. The first conjecture assumption A_1^{err} is shown in Figure 13.

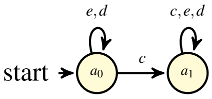


Figure 13: The first conjecture assumption A_1^{err}.

Now, we need to analyse if the first conjecture assumption A_1^{err} can be used to establish the AG reasoning proof rule, this is done through verification of the two promises of the AG (Feng et al., 2010):

- To verify $\langle true\rangle M_{WF_{TA}}\langle A\rangle_{\geq P_A}$, we first compute $M_{WF_{TA}} \parallel A_1^{err}$, then model checked $M_{WF_{TA}} \parallel A_1^{err}$ to obtain $1 - P_A$, the maximum probability of reaching the (undesirable) accepting states of A^{err}.
- To verify $\langle A\rangle_{\geq P_A} M_{WF_{AH}}\langle P\rangle_{\geq P_P}$, we first compute $M_{WF_{AH}} \parallel A_1^{err} \parallel P^{err}$. We used the multi-objective model checking algorithm to check the second promise because the satisfaction of A with probability P_A and of P with probability P_P must be analysed together. Theses steps are described in detail in (Feng et al., 2010). For our example we have used the latest version of PRISM, where probabilistic model checking algorithms and the multi-objective model checking algorithm are automated.

In practice, we start by checking the second promise, i.e, $\langle A\rangle_{\geq P_A} M_{WF_{AH}}\langle P\rangle_{\geq P_P}$, by using the model checker PRISM. This results an empty interval, which indicates that, even under the assumption $\langle A\rangle_{\geq 1}$, $M_{WF_{AH}}$ would still not satisfy P, therefore, we must refine A_1^{err}. To refine A_1^{err} we generate a probabilistic counterexample showing that $\langle A\rangle_{\geq 1} M_{WF_{AH}}\langle P\rangle_{\geq P_P}$ does not hold and then use this to generate traces for the L^* learning algorithm. More precisely, to generate this counterexample of $\langle A\rangle_{\geq 1} M_{WF_{AH}}\langle P\rangle_{\geq P_P}$, we first computed the parallel composition of $A_1^{err} \parallel M_{WF_{AH}} \parallel P^{err}$, then we have used COMICS[2] Tool to compute the minimal counterexamples for PAs. After few iterations, following the same process described in this example, our framework returns that the $M_{WF_{TA}} \parallel M_{WF_{AH}}$ satisfies the safety property P.

5 CONCLUSIONS

In this paper, we proposed an approach to formal modelling and verification of IOWF processes resulting from composition of two or more WF processes. The main aspects that characterise our approach are: (1) it supports both the analysis and the verification of both a standalone WF process schema or a composition of WF processes (IOWF), where each WF process is modelled using PA, (2) it supports the specification and verification of regular safety properties described in PCTL, (3) it covers all activities of a given WF process. To cope with the state space explosion problem faces when verifying large/complex IOWF, we proposed to use AG reasoning proof rule, and we have shown how to apply our approach using a simple example. We plan to extend this work by applying our approach to verify real life complex IOWF and use other AG reasoning rule such as (He et al., 2015). We moreover like to extend our translator to support BPMN description language.

REFERENCES

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web services : concepts, architectures and applications*. Springer.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106.

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*, volume 26202649. MIT press Cambridge.

Baier, C. and Kwiatkowska, M. (1998). Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155.

Bentahar, J., Yahyaoui, H., Kova, M., and Maamar, Z. (2013). Symbolic model checking composite web services using operational and control behaviors. *Expert Systems with Applications*, 40(2):508–522.

Braghetto, K. R., Ferreira, J. E., and Vincent, J.-M. (2011). Performance evaluation of business processes through a formal transformation to san. In *Computer Performance Engineering*, pages 42–56. Springer.

Calinescu, R., Kikuchi, S., and Johnson, K. (2012). Compositional reverification of probabilistic safety properties for large-scale complex it systems. In *Large-Scale Complex IT Systems. Development, Operation and Management*, pages 303–329. Springer.

Chen, Y.-F., Clarke, E. M., Farzan, A., Tsai, M.-H., Tsay, Y.-K., and Wang, B.-Y. (2010). Automated assume-guarantee reasoning through implicit learning. In *Computer Aided Verification*, pages 511–526. Springer.

Doshi, P., Goodwin, R., Akkiraju, R., and Verma, K. (2004). Dynamic workflow composition using markov decision processes. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 576–582. IEEE.

Etessami, K., Kwiatkowska, M., Vardi, M. Y., and Yannakakis, M. (2007). Multi-objective model checking of markov decision processes. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 50–65. Springer.

Feng, L. (2013). *On Learning Assumptions for Compositional Verification of Probabilistic Systems*. PhD thesis, University of Oxford.

Feng, L., Kwiatkowska, M., and Parker, D. (2010). Compositional verification of probabilistic systems using learning. In *Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the*, pages 133–142. IEEE.

Gabrel, V., Manouvrier, M., and Murat, C. (2013). A new linear program for qos-aware web service composition based on complex workflow.

Gallotti, S., Ghezzi, C., Mirandola, R., and Tamburrelli, G. (2008). Quality prediction of service compositions through probabilistic model checking. In *Quality of Software Architectures. Models and Architectures*, pages 119–134. Springer.

Gorton, S., Montangero, C., Reiff-Marganiec, S., and Semini, L. (2009). Stpowla: Soa, policies and workflows. In *Service-Oriented Computing-ICSOC 2007 Workshops*, pages 351–362. Springer.

Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535.

Hart, S. et al. (1984). Probabilistic temporal logics for finite and bounded models. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 1–13. ACM.

He, F., Gao, X., Wang, B.-Y., and Zhang, L. (2015). Leveraging weighted automata in compositional reasoning about concurrent probabilistic systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 503–514. ACM.

Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., et al. (2007). Web services business process execution language version 2.0. *OASIS standard*, 11(120):5.

Komuravelli, A., Pasareanu, C. S., and Clarke, E. M. (2012). Learning probabilistic systems from tree samples. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 441–450. IEEE Computer Society.

[2] http://www-i2.informatik.rwth-aachen.de/i2/comics/

Kwiatkowska, M., Norman, G., and Parker, D. (2009). Prism: probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45.

Larsen, K. G., Pettersson, P., and Yi, W. (1995). Compositional and symbolic model-checking of real-time systems. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 76–87. IEEE.

Lehmann, D. and Shelah, S. (1982). Reasoning with time and chance. *Information and Control*, 53(3):165–198.

Păsăreanu, C. S., Giannakopoulou, D., Bobaru, M. G., Cobleigh, J. M., and Barringer, H. (2008). Learning to divide and conquer: applying the l^* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205.

Sbaï, Z., Missaoui, A., Barkaoui, K., and Ben Ayed, R. (2010). On the verification of business processes by model checking techniques. In *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, volume 1, pages V1–97. IEEE.

Segala, R. (1995). *Modelling and verification of randomized distributed real time systems*. PhD thesis, Massachusetts Institute of Technology.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., and Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences*, 280:218–238.

Van Der Aalst, W. M. (1999). Process-oriented architectures for electronic commerce and interorganizational workflow. *Information systems*, 24(8):639–671.