Extending Model Checking to Data-Aware Temporal Properties of Web Services

Sylvain Hallé, Roger Villemaire, Omar Cherkaoui, Jérôme Tremblay, and Boubker Ghandour

Université du Québec à Montréal
C.P. 8888, Succ. Centre-ville
Montréal, Canada H3C 3P8
halle@info.uqam.ca

Abstract. A "data-aware" web service property is a constraint on the pattern of message exchanges of a workflow where the order of messages and their data content are interdependent. The logic CTL-FO$^+$ expresses these properties by allowing temporal operators and first-order quantification over message content to be freely mixed. A "naïve" translation of CTL-FO$^+$ into CTL leads to a serious exponential blow-up of the problem that prevents existing validation tools to be used. In this paper, we provide an alternate translation of CTL-FO$^+$ into CTL where the construction of the workflow model depends on the property to validate. We show experimentally how this translation is significantly more efficient and makes model checking of data-aware temporal properties on real-world web service workflows tractable using off-the-shelf tools.

1 Introduction

The phrase *web service validation* generally refers to the operation of checking the basic syntactical structure of the messages exchanged by a service for conformance to an interface description. It has long been known that to ensure a true interoperability of services, messages must also be sent and received in a proper sequence [2, 18]. Therefore, workflow validation extends to conformance to a set of temporal constraints; depending on the authors, the approach has been called "operating guidelines", "behavioural properties" or "protocol of interaction".

A large amount of works have studied this question from various angles, mostly borrowing from model checking techniques. The external behaviour of web services can be modelled by the transmission or reception of messages identified by propositional letters standing for their names [9, 12, 16, 22, 26, 35]. A possible refinement is to consider that the data exchanged in the messages of a web service can actually influence the control flow of that service [5,15,25,28,29]. A number of automated tools for the validation of the properties has been developed around this principle [8, 24, 30, 32]; most of them use standard model

We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada on this research.

checkers such as SPIN [23], NuSMV [10] or CWB [1] to validate the workflow models. The properties are expressed in Linear Temporal Logic (LTL), Computation Tree Logic (CTL), π-calculus, or a similar formalism [12, 32, 33].

These languages are called *propositional*: their atoms are propositional symbols over which first-order quantification is not allowed. For example, the CTL formula $\mathbf{AG}\,(a = x \rightarrow \mathbf{AF}\,b = x)$ correlates the values of state variables a and b at two different moments in time; it is a valid CTL formula when x is a static constant, but it cannot be used to express the same thing "for all x" unless the formula is repeated for every possible static value. This limited form of quantification is called *explicit*.

In contrast, there exist constraints where the sequence of messages and the data inside these messages are interdependent in such a way that first-order quantification is necessary; we call these properties "data-aware"; this concept was first introduced in [20]. We briefly show in Section 2 how these constraints arise naturally in a real-world web service scenario and are essential to validate. In Section 3, we present CTL-FO$^+$, a generalization of CTL that allows general first-order quantifiers to be freely mixed with temporal operators to express complex data-aware constraints. We show how it distinguishes itself from the few other methodologies suggested to model data-awareness; in particular, CTL-FO$^+$ model checking is decidable and PSPACE-complete.

There exist numerous ways to transform the CTL-FO$^+$ model checking problem back into classical CTL model checking to leverage existing workflow tools and standard model checkers; explicit quantification is one of them. Unfortunately, any such transformation results in an exponential blowup and shifts the original problem to the higher EXPTIME-hard class, unless P = NP. This result seems to suggest that data-aware properties are out of reach of existing tools.

However, in Section 4, we present a reduction of CTL-FO$^+$ to CTL that modifies the translation of a workflow into a finite-state system using the concept of "freeze quantification": the construction of the system becomes dependent on the property to validate. In Section 5, we compare this freeze quantification approach with the explicit quantification suggested above. Although both translations are ultimately exponential, we empirically demonstrate that freeze quantification is several orders of magnitude more efficient. We illustrate our claim by showing a technology chain using two off-the-shelf tools, the VERBUS [4] workflow translator coupled with the NuSMV [10] model checker to validate constraints on sample web service workflows. We conclude that despite the theoretical lower bound, it is nevertheless possible to model and validate data-aware properties in web services using existing technologies and a suitable reduction to CTL.

2 Data-Aware Web Service Properties

Our concern about capabilities of existing solutions to validate data-aware constraints comes from our observation that there exist real-world scenarios where such properties arise naturally.

We take as an example the User-Controlled Lightpath (UCLP) research project initiated by the CANARIE Consortium[1], which develops an environment that allows end users to self-provision and dynamically reconfigure optical network resources, called lightpaths, within a single domain or across multiple independent management domains. To this end, network resources from a specific provider are virtualized and exposed to the end user as instances of web services that implement functionality related to lightpath manipulation. Simply put, a *lightpath object* (LPO) is a point-to-point, high-speed optical link. In the UQAM-UO UCLP framework, each LPO is identified by a unique ID. The UCLP operations usually manipulate these IDs.

There are two main operations provided to manage LPOs. In order to build an end-to-end link, two adjacent LPOs can first be *concatenated*. The result of the concatenation operation is an LPO that is considered as one single link. In a dual manner, an LPO's bandwidth can be *partitioned* into links of equal bandwidth. This operation takes as input the reference to an LPO and returns an array of references to spawned lightpaths, each of the desired bandwidth. Typical partition and concatenate request and response XML messages are shown in Table 1.

```
<message>
  <operation>
    concatenateRequest
  </operation>
  <LPO-ID>i1</LPO-ID>
  <LPO-ID>i2</LPO-ID>
  ...
  <LPO-ID>in</LPO-ID>
</message>
```

```
<message>
  <operation>
    concatenateResponse
  </operation>
  <LPO-ID>i</LPO-ID>
</message>
```

```
<message>
  <operation>
    partitionRequest
  </operation>
  <LPO-ID>i</LPO-ID>
  <bandwidth>b</bandwidth>
  <login>ℓ</login>
</message>
```

```
<message>
  <operation>
    partitionResponse
  </operation>
  <LPO-ID>i1</LPO-ID>
  <LPO-ID>i2</LPO-ID>
  ...
  <LPO-ID>in</LPO-ID>
</message>
```

Table 1. The concatenate request, concatenate response, partition request and partition response XML messages.

Once lightpaths are exposed as web services, operations for manipulating them can be called like any other web service invocation in a process expressed

[1] http://www.canarie.ca/canet4/uclp/. The software developed by all CANARIE funded development teams is freely available from their site http://www.uclpv2.ca.

in a workflow language such as BPEL. However, these operations cannot be called arbitrarily.

To illustrate our point, we take the example of the partition operation, which takes as input the ID of some LPO x and returns new LPOs y, z corresponding to the results of the partition. It does not make sense to use x as an argument of a subsequent UCLP operation such as concatenate: although the LPO still physically exists, it has been logically superseded by its fragments y, z. The process could even have applied further operations on y and z, like concatenating them to other LPOs or further partitioning them. In this context, invoking, for example, a partition operation with x is at best semantically unsound and at worst plain dangerous for the reliability of the whole UCLP environment. We must therefore enforce the following constraint on any UCLP process:

UCLP Service Constraint 1 *Any LPO ID appearing in any partition request must be different from any LPO ID appearing in any future concatenate request.*

In that sentence, the first and third occurrences of the word "any" indicate a quantification over message content, while the second and fourth occurrences represent a quantification over messages in an execution sequence: data and temporal modalities are intertwined and the constraint is data-aware. Other data-aware constraints of technical nature can be easily found. We mention two of them which will be referred to in Section 5:

UCLP Service Constraint 2 *If two LPOs are the result of the same partition response, they cannot be involved together in any subsequent concatenate request.*

UCLP Service Constraint 3 *Every LPO occurring as an input of the concatenate operation must be of the same bandwidth.*

More details about UCLP and data-aware properties can be found in [20].

3 Formalizing Data-Aware Properties with CTL-FO$^+$

The previous properties express constraints on the data and sequentiality of messages exchanged by a workflow. Therefore, a suitable representation of this pattern of messages is a Kripke structure which contains state variables that represent the content of the messages that are sent or received. Discarding intermediate states where no message is received or sent, a path in the system corresponds to a possible sequence of messages in a service interaction. Properties about message sequences become properties on sequences of states that can then be expressed using temporal logics.

3.1 Syntax and Semantics of CTL-FO$^+$

The Computation Tree Logic with Full First-order Quantification (CTL-FO$^+$) is an extension of the well-known temporal logic CTL [11] aimed at describing sequentialities in a finite-state system while allowing full quantification over values of its state variables. Its syntax is defined as follows:

Definition 1 (Syntax) *The language CTL-FO$^+$ (Computation Tree Logic with general first-order quantification) is obtained by closing CTL under the following construction rules:*

1. *If x is a variable or a constant, and y is either a variable, a constant or a state variable, then $x = y$ is a CTL-FO$^+$ formula;*
2. *If φ and ψ are CTL-FO$^+$ formulæ, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, $\mathbf{AG}\,\varphi$, $\mathbf{EG}\,\varphi$, $\mathbf{AF}\,\varphi$, $\mathbf{EF}\,\varphi$, $\mathbf{AX}\,\varphi$, $\mathbf{EX}\,\varphi$, $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$, $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$, are CTL-FO$^+$ formulæ;*
3. *If φ is a CTL-FO$^+$ formula and x is a free variable in φ, then $\exists x : \varphi(x)$ and $\forall x : \varphi(x)$ are CTL-FO$^+$ formulæ.*

As usual [11], semantics can be defined in terms of the adequate set of operators $\exists$, **AF** , **EX** and **EU** , $\neg$ and $\vee$:

Definition 2 (Semantics) *Let $K = (S, I, R, L)$ be a Kripke structure, with S the set of states, I the set of initial states, $R \subseteq S^2$ the transition relation and L a labelling of states. Let $s_0 \in S$ be a state. Define a path $\pi = s_0, s_1, \ldots$ as a sequence of states in S such that $(s_i, s_{i+1}) \in R$ for every $i \geq 0$. Let $D_s(x)$ be the (finite) set of possible values for a quantified variable x in state s, p be some state variable in K and c_1 and c_2 be constants. We say the pair K, s_0 satisfies the CTL-FO$^+$ formula φ if and only if it respects the following rules:*

$$
\begin{aligned}
K, s_0 \models p = c_1 &\Leftrightarrow p \text{ is equal to } c_1 \text{ in state } s_0 \\
K, s_0 \models c_1 = c_2 &\Leftrightarrow c_1 \text{ is equal to } c_2 \\
K, s_0 \models \neg\varphi &\Leftrightarrow K, s_0 \not\models \varphi \\
K, s_0 \models \varphi \vee \psi &\Leftrightarrow K, s_0 \models \varphi \text{ or } K, s_0 \models \psi \\
K, s_0 \models \mathbf{AF}\,\varphi &\Leftrightarrow \text{for each } \pi = s_0 s_1 s_2 \ldots, K, s_i \models \varphi \text{ for some } i \\
K, s_0 \models \mathbf{EX}\,\varphi &\Leftrightarrow \text{there exists } \pi = s_0 s_1 s_2 \ldots \text{ such that } K, s_1 \models \varphi \\
K, s_0 \models \mathbf{E}\,\varphi\,\mathbf{U}\,\psi &\Leftrightarrow \text{there exists } \pi = s_0 s_1 s_2 \ldots \text{ such that } K, s_j \models \psi \text{ for some } j \\
&\qquad \text{and } K, s_i \models \varphi \text{ for } i < j \\
K, s_0 \models \exists x\,\varphi(x) &\Leftrightarrow \text{there exists } a \in D_{s_0}(x) \text{ such that } K, s_0 \models \varphi(a)
\end{aligned}
$$

By extension, we write $K \models \varphi$ if the initial state s of K is such that $K, s \models \varphi$.

The values of the variables appearing in a CTL-FO$^+$ formula are quantified according to specific elements of the XML message that is received or sent in the current state of the system. To indicate this, we add a subscript to the quantifier indicating the name of the element. A quantifier like $\forall_{\text{LPO-ID}} x$ therefore means "for all values x of elements named LPO-ID in the current message". Therefore, the domain of a variable depends on the content of the current message, which in turn depends on the current state of the system. By extension, we write $D(x)$ to designate the union of the $D_s(x)$ for all $s \in S$.

Using such notation, UCLP Service Constraint 1 becomes the following CTL-FO$^+$ formula:

UCLP Formal Service Constraint 1

$$\begin{aligned}&\mathbf{AG}\,(\forall_{\mathit{operation}}\, x_1 : x_1 = \mathit{concatenateRequest} \rightarrow \\ &\forall_{\mathit{LPO\text{-}ID}}\, x_2 : \mathbf{AX}\ \mathbf{AG}\ (\forall_{\mathit{operation}}\, x_3 : \\ &\quad x_3 = \mathit{partitionRequest} \rightarrow \forall_{\mathit{LPO\text{-}ID}}\, x_4 : x_2 \neq x_4))\end{aligned}$$

This formula states that at any time in any execution of the process, if the operation x_1 of the message is concatenateRequest, then for every LPO ID x_2 appearing in this message, we have that for every future message whose operation element value x_3 is partitionRequest, any value x_4 for its LPO ID is different from x_2. In other words, once an LPO has been concatenated, no further partition involves this LPO, which is indeed equivalent to UCLP Service Constraint 1. UCLP Service Constraints 2 and 3 can be formalized into similar CTL-FO$^+$ formulæ.

We are only aware of a limited number of works related to data-awareness in temporal properties. CTL-FO$^+$ is reminiscent of EQCTL that extends CTL by allowing existential quantification over *state variables* [27]. EQCTL is not closed under negation; therefore, universal quantification cannot be obtained; moreover, CTL-FO$^+$ quantifies over *values* and is closer to true first-order quantification. QCTL [31] extends CTL by including first-order quantification and monadic second-order quantification over arbitrary *algebraic data structures*. Such expressiveness is not required in our case. Specifications using XQuery on traces (SXQT) are defined in [34], but the approach allows the validation of one specific trace at a time; graph transformation rules [21] allow the description of data modifications but lack the ability to express temporal modalities; a logic called CTL-FO is introduced in [14], but its general model checking problem is shown to be undecidable. CTL-FO$^+$ generalizes CTL-FO by allowing free use of the existential quantifier.

3.2 Model Checking CTL-FO$^+$ Properties

Now that we have shown how data-aware properties can be expressed in CTL-FO$^+$, the next question is how to efficiently perform the model checking of these formulæ. We first establish the complexity of the problem with the following theorem.

Theorem 1 *Let $K = (S, I, R, L)$ be a Kripke structure modelling a particular web service and φ be a CTL-FO$^+$ formula. The problem of deciding whether $K \models \varphi$ is PSPACE-complete.*

Proof. PSPACE-hardness is obtained by reduction to the Quantified Boolean Formula (QBF) problem [17]; it suffices to observe that a QBF is by definition a CTL-FO$^+$ formula. A model checking algorithm can be devised in a straightforward manner from the classical CTL model checking algorithm. This algorithm performs a structural recursion on the formula and computes a set of states depending on the top-level operator. It suffices to add an additional case to the

algorithm when the top-level operator is a quantifier of the form $\exists x\, \varphi(x)$. The algorithm simply calls itself on $\varphi(x)$ for every possible value of the variable x and computes the union of all sets of states returned by each such call. Every recursive call of the algorithm returns a subset of S and the height of the stack is bounded by $|\varphi|$, the length of φ. Since domains are considered finite, the total space consumed is polynomial in $|\varphi|$. □

Although CTL-FO$^+$ is a generalization of CTL-FO mentioned above, we apply it on a simpler model of workflows that makes its model checking decidable. In particular, since all UCLP properties only involve equality between values, finitely many symbols are needed to handle infinite domains. This result shows that CTL-FO$^+$ is a generalization of CTL, whose model checking is in P. Therefore, existing web service tools and model checkers cannot be used as is to validate data aware properties. However, one can easily use the semantics definition of the existential operator and explicitly enumerate all possible values $k_1, k_2, \ldots, k_n$ in the domain D of the quantified variable. The universally quantified formula $\forall x : \mathbf{AG}\,(a = x \rightarrow \mathbf{AF}\, b = x)$ hence becomes:

$$\begin{aligned}&\mathbf{AG}\,(a = k_1 \rightarrow \mathbf{AF}\, b = k_1) \wedge \mathbf{AG}\,(a = k_2 \rightarrow \mathbf{AF}\, b = k_2)\\ &\quad \wedge \cdots \wedge \mathbf{AG}\,(a = k_n \rightarrow \mathbf{AF}\, b = k_n)\end{aligned}$$

The resulting expression is a plain CTL formula where all references to data are now static, which amounts to extending the message alphabet. This approach has already been suggested in Section 1 and has been called "explicit quantification". However, the next theorem shows that this construction is unlikely to be optimal.

Theorem 2 *If there exists a polynomial reduction of CTL-FO$^+$ model checking to CTL model checking, then P = NP.*

Proof. Suppose that that for every Kripke structure K and every CTL-FO$^+$ formula φ, there exists a polynomial translation of K into a Kripke structure K' and a polynomial translation of φ into a CTL formula φ'. Since CTL model checking is in P [11], and that P $\subseteq$ PSPACE [17, sect. 7.4], then from Theorem 1 PSPACE $\subseteq$ P, which can be true only if P = NP. □

4 An Efficient Reduction of CTL-FO$^+$ to CTL

Theorem 2 seems to indicate that in fact, *any* attempt to use standard model checkers to validate data-aware workflow properties is "doomed" to an exponential blow-up of the original problem, and not only the explicit quantification method suggested above. In this section, we show an alternate translation of the CTL-FO$^+$ model checking problem to CTL which, while still in EXPTIME, performs much better.

The method employed to achieve this result uses a technique called "freeze quantification" where additional variables can be added to a Kripke structure that can be used to "freeze" the value of a state variable at some point in

the execution for future reference. It has been originally developed in [3] for timed transitions systems and further studied in [13]. [19] used this technique to reduce a subset of XPath to CTL. We proceed in two steps: first, we show how to convert a Kripke structure $K = (S, I, R, L)$ and a CTL-FO$^+$ formula φ to a Kripke structure $K_\varphi = (S', I', R', L')$; then, we show how a CTL-FO$^+$ formula φ can be translated to a CTL formula φ' and show that φ is true for K if and only if φ' is true for K_φ, thereby reducing the problem of CTL-FO$^+$ model checking to CTL model checking.

4.1 Transforming a Kripke Structure

Set of system states. The principle consists in adding to the original Kripke structure one system variable for each distinct quantified variable appearing in the CTL-FO$^+$ formula to validate. These variables are called the "freeze" variables, since they are intended to capture the value of some part of a message at a given point in the execution of the workflow. For example, to validate UCLP Formal Service Constraint 1, four additional variables are needed corresponding to the variables x_1 to x_4 in the CTL-FO$^+$ formula. At the start of any execution sequence, these variables take a special value noted # that indicates they have not yet taken any "real" value.

The domain of each freeze variable is dependent on the message part on which they are defined; in the previous example, the freeze variable x_1 is defined on `operation` elements; its domain is the set of all values appearing inside such elements somewhere in the process. In contrast, the freeze variable x_2 is defined on elements of name `LPO-ID`; its domain is the set of all possible values that can occur in this part of any message during the process. The computation of the domain of each variable is an easy task that can be statically computed on the original BPEL process; most web service validation tools can perform it automatically.

Formally, if we let $D(x_1), D(x_2), \ldots, D(x_n)$ be the respective domains of freeze variables $x_1, x_2, \ldots, x_n$, the set S' of states in K_φ is defined as $S' = S \times D(x_1) \times D(x_2) \ldots D(x_n)$. That is, the new Kripke structure is simply the extension of the original system to the n freeze variables. Consequently, we say that state $s' \in S'$ is an *extension* of $s \in S$ if all the non-freeze variables have the same values in both states; conversely, s is the (unique) *restriction* of s'.

The initial state of K_φ is defined uniquely by choosing the state s for which all non-freeze variables are identical as in the initial state of K, and where all freeze variables take the value #.

Transition relation. We then define the transition relation of K_φ in the following way. In all states of the system where no message is sent or received, the transitions are left untouched. The internal variables of the model can change value in the same way as in the original Kripke structure, while all the freeze variables do not change. That is, if $s, t \in S$, $(s', t') \in S'$ and s' and t' are respective extensions of s and t such that all freeze variables have the same values in both s' and t', and s is a state where no BPEL communicating activity (`invoke`, `receive` or `reply`) occurs, then $(s, t) \in R$ if and only if $(s', t') \in R'$.

Let us now consider the case of a state of the model where a message is either sent or received. Such a state is exploded in the resulting Kripke structure into two phases: in the first phase, the non-freeze variables can change according to the original transition relation as described previously, while the freeze variables stay the same. Once these changes are made, the system enters into a "freezing" phase where the roles are reversed: the non-freeze variables keep their values, and some freeze variables can change in a specific way.

In the freezing phase, a freeze variable can stay undefined, or take the value of some part of the *current* message. It is important to remark that the variables can only be assigned values corresponding to the message part on which they are defined. Consider for example the sequence formed of a partitionRequest and a partitionResponse XML messages as shown in Table 1. In the first message of the pattern, variable x_1 can only take the value "partitionRequest", since x_1 is defined in UCLP Formal Service Constraint 1 as a variable on root element names. In the same way, x_2 can only take the value i, since x_2 is defined as a variable on elements of name <LPO-ID>. In the second message, x_2 can take the values $i_1, \ldots, i_n$.

The actual value assigned to either of these special variables in each state is non-deterministic. In addition, each variable may or may not take a value –that is, variables can stay undefined. However, once a variable has taken a defined value, it keeps this value for the remainder of the execution trace.

The exit from the freezing phase is also non-deterministic. The system loops any number of times into the freezing phase of a given state and then comes out and resumes its execution as specified by the original transition relation.

Formally, let $s \in S$ be a state of the model where an `invoke`, `receive` or `reply` occurs and $s' \in S'$ be an extension of s. Let x_i be a freeze variable and $v \in D_s(x_i)$. Let $t \in S$ be a state such that $(s,t) \in R$ and $t' \in S'$. Then $(s',t') \in S'$ if and only t' is an extension of s or an extension of t, $x_i = \#$ in s', $x_i = v$ in t' and for all $1 \leq j \leq n$, if $i \neq j$, x_j does not change between s' and t'. We must then show that these modifications preserves the behaviour of the original model.

Theorem 3 *Let π' be an execution in K_φ, and let π'' be the sequence of states in S obtained by taking the reduction of π'. There exists an execution π in K such that π'' and π are stuttering equivalent.*

Proof. By construction, every transition (s'_1, s'_2) in K_φ that does not enter or leave a freezing phase is the extension of some transition (s_1, s_2) in K. It remains to observe that any sequence of states $t_1, \ldots, t_i$ in a freezing phase in K_φ leaves all non-freeze variables unchanged; therefore, the reduction of that sequence is nothing but the repetition of the same state in K, i times. It follows that π'' is the same sequence of states as some execution π in K, with the possible exception that some states are repeated a finite number of times, thus leading to stuttering equivalence. □

4.2 Converting a CTL-FO$^+$ Formula

Once the Kripke structure has been modified in the previously described manner, the conversion of a CTL-FO$^+$ formula into a standard CTL formula becomes straightforward.

We define a linear embedding ω of CTL-FO$^+$ into CTL formulæ. Let φ and ψ be CTL-FO$^+$ formulæ, c be a constant in V, m and n be message part names in N, and the x_i be quantified variables in the CTL-FO$^+$ formula. Translating the Boolean connectives and the ground equality testings is direct.

The quantification on variables becomes a quantification on some execution paths. In effect, a quantifier like $\forall x : \varphi(x)$ actually means "for all possible values x can take in the current state, $\varphi(x)$ holds". According to the Kripke structure K_φ defined previously, this simply amounts to asserting that in the current state, for all possible ways for x of changing from # to some definite value, φ is true, which becomes the following translation:

$$\omega(\forall_n x_i : \varphi) \equiv (x_i = \# \rightarrow \mathbf{AX}\,(x_i \neq \# \rightarrow \omega(\varphi))) \tag{1}$$

Similarly, a quantifier like $\exists x : \varphi(x)$ actually means "there exists a value x can take in the current state such that $\varphi(x)$ holds". According to the Kripke structure K_φ defined previously, this simply amounts to asserting that in the current state, there exists a way for x of changing from # to some definite value where φ is true. This becomes the following translation:

$$\omega(\exists_n x_i : \varphi) \equiv (x_i = \# \rightarrow \mathbf{EX}\,(x_i \neq \# \wedge \omega(\varphi))) \tag{2}$$

The translation of the CTL temporal operators is also direct, except for the next quantifiers **AX** and **EX**. The next state state in K is not necessarily the next state in K_φ because of the possible freeze loops that may put two consecutive states in K arbitrarily far apart in K_φ. However, it is possible to work around this by asserting that the next state in K is mirrored by the next *non-freezing* state in K_φ. We obtain:

$$\omega(\mathbf{AX}\,\varphi) \equiv \mathbf{A}\,\gamma\mathbf{U}\,\omega(\varphi) \tag{3}$$

$$\omega(\mathbf{EX}\,\varphi) \equiv \mathbf{E}\,\gamma\mathbf{U}\,\omega(\varphi) \tag{4}$$

where γ is an additional Boolean variable that is true whenever the system is in a freezing phase, and false otherwise. Formula (3) hence asserts that in K_φ, as soon as the system gets out of the current freezing phase (if any), φ must be true.

Using this embedding, UCLP Formal Constraint 1 is recursively translated to the following CTL expression.

$$
\begin{aligned}
&\mathbf{AG}\,(x_1 = \# \rightarrow (\mathbf{AX}\,(x_1 \neq \# \rightarrow \\
&\ (x_2 = \# \rightarrow (\mathbf{AX}\,(x_2 \neq \# \rightarrow \\
&\quad x_1 = \text{partitionRequest} \rightarrow \\
&\quad \mathbf{AX}\,\mathbf{AG}\,(x_3 = \# \rightarrow (\mathbf{AX}\,(x_3 \neq \# \rightarrow \\
&\quad\ (x_4 = \# \rightarrow (\mathbf{AX}\,(x_4 \neq \# \rightarrow \\
&\quad\quad x_3 = \text{concatenateRequest} \rightarrow x_2 \neq x_4))))))))))))
\end{aligned}
\tag{5}
$$

We do not expect data aware constraints to be expressed directly in CTL in such a way. However, the translation from CTL-FO$^+$ to CTL can be automated, and the next theorem shows that the overall construction preserves the validity of the original problem.

Theorem 4 *Let K be a Kripke structure, φ be a CTL-FO$^+$ formula, K_φ be a converted Kripke structure and $\varphi' = \omega(\varphi)$. Then φ is true for K if and only if φ' is true for K_φ.*

We only sketch the proof here, which is done by induction on the structure of the formula. By Theorem 3, we know that reachability of states is preserved when converting K to K_φ. It remains to show that an assignment ρ of values to the freeze variables $x_1, \ldots, x_n$ is true for φ in K if and only if it is true for φ' in K_φ, which can be realized by observing the definition of the freeze transitions.

Contrarily to explicit quantification, the freeze quantification approach does not cause an exponential blow-up of the original formula. The embedding ω is linear: that is, if we denote by $|\varphi|$ the length of a CTL-FO$^+$ formula φ, then $|\omega(\varphi)| \in O(|\varphi|)$. It suffices to remark that each translation rule consumes at least one symbol of the original CTL-FO$^+$ formula and contributes a fixed number of symbols in the resulting CTL formula.

5 Experimental Results

To confirm the soundness of our approach, we conducted a set of experiments that involved the validation of UCLP constraints detailed in Section 2 on sample BPEL processes taken from real-world UCLP use cases. The goal of these experiments was to show that validating UCLP constraints can be effectively done using the freeze quantification solution presented in this paper. Furthermore, we also show that the explicit model checking approach quickly becomes inadequate for these properties.

5.1 Methodology

The experiments were made using only readily available and open source tools. NuSMV [10] was chosen as the model checker because of its robustness, good performance, and especially its capability of model checking CTL formulæ. VERBUS [4] was chosen to generate the Kripke structure from the original BPEL

processes; the choice was influenced mostly by its ability to model the content of variables and messages in a process and its capability to directly output the Kripke structure as an SMV file. Only minor bug-solving modifications were made to the original code of VERBUS to make it work in our situation. The modified version of VERBUS, a compiled copy of NuSMV, the BPEL modification routines and all the files used in the experiments are released under the GNU GPL.[2]

The SMV files produced by VERBUS were then modified according either to the explicit quantification or the freeze approach. In the explicit quantification, no modification other than appending the desired CTL formula at the end of the file was made. In the freeze approach, freeze variables and freeze transitions were added throughout the file, and the corresponding CTL formula was also added at the end.

5.2 Results and Discussion

We then proceeded to this method successively with sample BPEL processes. Each process consisted in one concatenation and one partition of a given number of LPOs. For $n = 1$, the result of all operations in the process is deterministic: for example, the partition operation with LPO A always returns the same LPO IDs B, C, D. We then varied this number n of LPOs by making the operations non-deterministically return IDs taken from a set of possible values. This generalization of the process is a natural step to take, since a UCLP operation on an LPO is dependent of the global situation of the UCLP network and therefore need not return the same value in every invocation.

The validation times for the freeze and the explicit quantification approaches are presented in Table 2, for each UCLP formal constraint, for BPEL processes with n ranging from 1 to 4. Similarly, the size of the NuSMV file required to validate the process is shown in Table 3. All times have been obtained with NuSMV 2.4.0 on an AMD Athlon 2200+ CPU running under Windows XP. Since NuSMV takes several dozens of seconds only to display the explicitly quantified formulæ, this time was not included in the results.

	Freeze quantification					Explicit quantification				
Constraint	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
1	1.5	1.9	4.5	5.4	8.7	0.2	0.3	0.4	0.4	0.5
2	1.6	2.2	4.4	5.4	9.2	13	36	101	222	884
3	1.8	2.2	4.6	5.8	9.3	—	—	—	—	—

Table 2. Validation time (in seconds) of sample BPEL processes with UCLP constraints, using respectively the freeze and the explicit quantification approach.

[2] All the material is available from
http://www.teleinfo.uqam.ca/Members/halle_sylvain/uclp

	Freeze quantification					Explicit quantification				
Constraint	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
1	30.3	30.5	30.6	30.7	30.8	237	299	368	445	529
2	33.9	34.0	34.2	34.3	34.5	10542	18156	29249	47939	70509
3	34.8	35.0	35.1	35.3	35.5	111597	198831	329075	—	—

Table 3. File size (in kilobytes) of the NuSMV files for the validation of sample BPEL processes with UCLP constraints, using respectively the freeze and the explicit quantification approach.

These figures confirm what was suggested in Section 3.2: using explicit quantification to produce the CTL formula quickly takes its toll on the validation time. As expected, the translation of the CTL-FO$^+$ formula is heavily dependent on n and grows exponentially. With UCLP operations returning only 5 different possible values, validation time takes almost 15 minutes. We could not get any times for UCLP Formal Constraint 3 since NuSMV crashed before reaching the end of the files, most probably due to their size. We did not dare to generate the formulæ for $n = 4$ and $n = 5$ that were expected to occupy more than 500 Mb each. Comparatively, the freeze approach requires only minimal modifications to the original NuSMV file produced by VERBUS; most importantly, these modifications are constant and do not depend on n. The increase in the file size is only due to the addition of the non-deterministic transitions required when incrementing n. Validation time, no matter the property, exhibits the same behaviour and grows much more reasonably: the 884 seconds required to validate property 2 with $n = 5$ in the explicit quantification is improved by a factor 80 and falls to less than 10 seconds using the freeze approach.

These trends do not hold for UCLP Formal Service Constraint 1 where the explicit quantification approach fares better than the freeze approach. This can be explained by the fact that the addition of freeze variables and transitions imposes an initial overhead on the Kripke structure that the explicit quantification does not have. Since in this situation, the CTL formulæ to validate are only a few hundred kilobytes long, they can be considered too small to represent a serious load on the model checker.

6 Conclusions

In this paper, we have shown that the composition of User-Controlled Lightpath resources is subject to constraints involving both the sequentiality of messages and the content of these messages. We have demonstrated how current traditional model checking approaches to the validation of web service workflows are insufficient for the validation of UCLP scripts subject to these kinds of constraints. We have demonstrated by empirical testing on real-world BPEL processes how an extension of CTL called CTL-FO$^+$ can be used to effectively model these complex workflow properties; we showed how a suitable reduction of CTL-FO$^+$

to CTL can be used to validate them in reasonable time compared to classical approaches.

The success of this project opens the way for future developments. First, we plan to integrate the automated generation of freeze variables and transitions directly into VERBUS. Second, it is possible to expand the range of expressiveness of CTL-FO$^+$ formulæ by taking into account not only the values inside XML messages exchanged by a BPEL process, but also their hierarchical organization into trees, thereby embedding into CTL-FO$^+$ a subset of a tree language like XPath.

References

1. The Edinburgh concurrency workbench.
2. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services, Concepts, Architectures and Applications*. Springer, 2004.
3. R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.
4. J. Arias-Fisteus, L. S. Fernández, and C. D. Kloos. Applying model checking to BPEL4WS business collaborations. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 826–830. ACM, 2005.
5. D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In Böhm et al. [6], pages 613–624.
6. K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors. *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. ACM, 2005.
7. M. Bravetti, M. Núñez, and G. Zavattaro, editors. *Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings*, volume 4184 of *Lecture Notes in Computer Science*. Springer, 2006.
8. A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web service choreographies. *Electr. Notes Theor. Comput. Sci.*, 105:73–94, 2004.
9. T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW*, pages 403–410, 2003.
10. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.
11. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
12. G. Decker, J. M. Zaha, and M. Dumas. Execution semantics for service choreographies. In Bravetti et al. [7], pages 163–177.
13. S. Demri, R. Lazic, and D. Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. In *TIME*, pages 113–121. IEEE Computer Society, 2005.
14. A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. In A. Deutsch, editor, *PODS*, pages 71–82. ACM, 2004.
15. Z. Duan, A. J. Bernstein, P. M. Lewis, and S. Lu. A model for abstract process specification, verification and composition. In M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors, *ICSOC*, pages 232–241. ACM, 2004.

16. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based analysis of obligations in web service choreography. In *AICT/ICIW*, page 149. IEEE Computer Society, 2006.
17. M. R. Garey and D. S. Johnson. *Computers and intractability, a guide to the theory of NP-completeness*. W. H. Freeman, 1979.
18. P. Greenfield, D. Kuo, S. Nepal, and A. Fekete. Consistency for web services applications. In Böhm et al. [6], pages 1199–1203.
19. S. Hallé, R. Villemaire, and O. Cherkaoui. CTL model checking for labelled tree queries. In *TIME*, pages 27–35. IEEE Computer Society, 2006.
20. S. Hallé, R. Villemaire, O. Cherkaoui, and B. Ghandour. Model-checking data-aware temporal workflow properties with CTL-FO+. In *EDOC*, pages 267–278. IEEE Computer Society, 2007.
21. R. Heckel and L. Mariani. Automatic conformance testing of web services. In M. Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2005.
22. S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri nets. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management*, volume 3649, pages 220–235, 2005.
23. G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.
24. J. E. Johnson, D. E. Langworthy, L. Lamport, and F. H. Vogt. Formal specification of a web services protocol. *Electr. Notes Theor. Comput. Sci.*, 105:147–158, 2004.
25. R. Kazhamiakin, M. Pistore, and L. Santuari. Analysis of communication models in web service compositions. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 267–276. ACM, 2006.
26. M. Koshkina and F. van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *ACM SIGSOFT SEN*, 29(5), September 2004.
27. O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. In P. Wolper, editor, *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 325–338. Springer, 1995.
28. N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting BPEL processes. In S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, editors, *BPM*, volume 4102 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2006.
29. S. Nakajima. Model-checking of safety and security aspects in web service flows. In N. Koch, P. Fraternali, and M. Wirsing, editors, *ICWE*, volume 3140 of *Lecture Notes in Computer Science*, pages 488–501. Springer, 2004.
30. M. Pistore, M. Roveri, and P. Busetta. Requirements-driven verification of web services. *Electr. Notes Theor. Comput. Sci.*, 105:95–108, 2004.
31. A. Rensink. Model checking quantified computation tree logic. In C. Baier and H. Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2006.
32. K. J. Turner. Formalising web services. In F. Wang, editor, *FORTE*, volume 3731 of *Lecture Notes in Computer Science*, pages 473–488. Springer, 2005.
33. W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. In Bravetti et al. [7], pages 1–23.
34. M. Venzke. Specifications using XQuery expressions on traces. *Electr. Notes Theor. Comput. Sci.*, 105:109–118, 2004.
35. J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, and G. Decker. Service interaction modeling: Bridging global and local views. In *EDOC*, pages 45–55. IEEE Computer Society, 2006.