Abstract
In this paper, a method for modeling and analyzing workflows based on Temporal Logic of Actions is presented. After a brief review of TLA, present the semantic of workflow process using TLA and illustrate how to modeling and analysis workflow using TLA. In TLA, both the workflow model and its properties are modeled by TLA formulas. Analysis of workflow model in TLA is carried out by validating the implication relationship between two formulas. Finally, the process of modeling and analysis is validated through a case study.

1. Introduction
A workflow is a collection of cooperating, coordinated activities designed to carry out a well-defined complex process, such as trip planning, graduate student registration procedure, or a business process in a large enterprise. An activity in a workflow might be performed by a human, a device, or a program. Workflow management system is a framework for capturing the interaction among the activities in a workflow and is recognized as a new paradigm for integrating disparate systems, including legacy systems. Ideally, they should also help the user in analysis and reasoning about complex business processes. Despite many workflow management systems developed for different types of workflow based on different paradigms, the lack of rigorous theoretic foundation and then effective model analysis methods has blocked workflow techniques’ research and application [1]. The method of analysis of workflow depends on the method of modeling. The method of modeling workflow could be classified into formalization methods and non-formalization methods. Non-formalization methods, such as activity diagram, WPDL, and Speech Act, are lack in supporting the analysis. Among the formalization methods, modeling workflow by Petri nets is the most important one. And there also have some analysis methods based on the Petri nets. However, analysis of workflow based on Petri nets has its limitations:
- There is no unified framework to modeling and analysis of workflow based on Petri nets. In order to analysis different properties, workflow is modeled in different types of Petri nets and analyzed in ad hoc approaches. So if we want to analyze a new property of workflow, we need to model the workflow using a special type of Petri nets and develop a new approach of analysis.
- The existing methods could not help people in designing a workflow, which satisfies the given property.

In order to model and analyze workflow more efficiently, in this paper, we develop a framework for modeling and analyzing workflows based on Temporal Logic. In our framework, both the workflow model and its properties are modeled by TLA formulas. Analysis of workflow model in TLA is carried out by validating the implication relationship between two formulas. In recent years, various kinds of temporal logic emerge for modeling and analyzing real world systems. Temporal logic now has becoming not only an important tool of modeling of real word systems but also a tool of model analysis because of its ability of reasoning. The earliest attempts at formalizing a time calculus date back to 1941 by Findlay, and 1955 by Prior. Since then, there has been a number of attempts on issues related to this subject matter, like topology of time [2], first-order and modal approaches to time [3], treatments of time for simulating action and language, etc. Several attempts on mechanizing the temporal reasoning processes have also been reported in the literature [4]-[6]. A majority of these approaches is formulated to handle issues relevant to a specific application area of interest to the scholar(s) who proposed the approach. Therefore, each approach comes with its advantages in addressing certain issues and disadvantages for lack of generality in handling others. Artificial Intelligence, software engineering, planning, cognitive sciences, time-sensitive databases, program verification
and modeling of discrete-event systems are some of the examples of the application areas of temporal logic.

The rest of the paper is structured as follows. In Section 2, a brief review of Temporal Logic of Actions is provided. Section 3 present how to modeling workflow base on Temporal Logic of Actions. The section also presents how to analysis the model by using the reasoning in Temporal Logic of Actions. In section 4, we give a case study to illustrate the process of modeling and analysis of workflow model using Temporal Logic of Actions. Finally, the paper is concluded in section 5 with some concluding remarks and future directions.

2. Temporal Logic of Actions

The logic that we use for modeling and analysis of workflow is the Temporal Logic of Actions, abbreviated as TLA. This section presents a brief review of Temporal Logic of Actions. For completed specification of Temporal Logic of Actions, readers could refer to [7]. All TLA formulas can be expressed in terms of familiar mathematical operators (such as \( \land, \lor \)) plus four new ones: ' (prime), \( \Box \), \( \Diamond \), and \( \exists \). TLA combines two logics: Logic of Actions and a standard Temporal Logic. A logic consists of a set of rules for manipulating formulas. To understand what the formulas and their manipulation mean, we need a semantics. The semantics of our logic is defined in terms of behavior, state, and action. A behavior is a finite sequence of states. A state is an assignment of values to variables. Thus a state \( s \) assigns a value \( s(x) \) to a variable \( x \). The collection of all possible states is denoted \( St \). We write \( s[x] \) to denote \( s(x) \). An action is a Boolean-valued expression formed from variables, primed variables, and constant symbols. It presents the relationship between current state and the next state. Now we describe TLA itself and give its syntax, formal semantics, derived notation, axioms and proof rules.

Syntax of TLA:

- formula ::= predicate\( \land \)formul-formula\( \lor \)formul-formula\( \exists \)formula
- action ::= Boolean-valued expression formed from variables, primed variables, and constant symbols
- predicate ::= (action) with no primed variables | Enabled (action)
- state function ::= No Boolean-valued expression formed from variables, and constant symbols

Semantics of TLA:

\[
s[f] = f(\forall \nu v: s[\nu v]/v)
\]

\[
\sigma[F \land G] = \sigma[F] \land \sigma[G]
\]

\[
s[A|f] = A(\forall \nu v': s[\nu v]/v,f[v]/v')
\]

\[
\sigma[\neg F] = \neg \sigma[F]
\]

\[
s[Enabled A] := \exists t \in St : s[A]t
\]

\[
(s_0, s_1, s_2, \cdots) [A] := s_0[A] s_1
\]

\[
(s_0, s_1, s_2, \cdots) [\square F] := \forall n \in N : (s_n, s_{n+1}, s_{n+2}, \cdots) [F]
\]

Additional notation

\[
p' := p(\forall v : v'/v)
\]

\[
\Diamond F \iff \neg \Box \neg F
\]

\[
[A]_f := A \lor (f' \equiv f)
\]

\[
\langle A \rangle_f := A \land (f' \not\equiv f)
\]

\[
F \text{ lead to } G := \Box(F \Rightarrow \Diamond G)
\]

\[
WF(A) := \Box \diamond \langle A \rangle_f \land \Box \neg \Box \neg \text{ Enabled } \langle A \rangle_f
\]

\[
SF(A) := \Box \diamond \langle A \rangle_f \lor \Box \neg \Box \neg \text{ Enabled } \langle A \rangle_f
\]

Unchanged \( f \) := \( f' = f \)

Axioms of TLA:

1) \( \vdash (F \text{ provable by propositional logic}) \Rightarrow \Box F \)
2) \( \vdash (F \Rightarrow G) \Rightarrow (\Box F \Rightarrow \Box G) \)
3) \( \vdash \Box F \Rightarrow F \)
4) \( \vdash \Box F \equiv F \)
5) \( \vdash (\Box F \land G) \equiv (\Box F) \land (\Box G) \)
6) \( \vdash (\Box F) \land (\Box G) \equiv (\Box (F \land G) \land G) \)

Some proof rules of TLA:

1) \( \vdash (P \land (f' = f) \Rightarrow P') \)
2) \( \vdash (P \Rightarrow \Box F) \Rightarrow (\Box P \land [P \Rightarrow P'])_f \)
3) \( \vdash (I \land [N]_f \Rightarrow I') \)
4) \( \vdash \Box I \Rightarrow (\Box [N]_f \equiv \Box [N \land I \land I'])_f \)
5) \( \vdash (P \land [N]_f \Rightarrow (P \land Q)_f) \Rightarrow (P \land Q)_f \Rightarrow (\Box P \land \Box Q \land \Box Enabled \langle A \rangle_f) \Rightarrow (\Box [N]_f \land P \Rightarrow Q) \Rightarrow P \land \Box Enabled \langle A \rangle_f \)

Where \( s, s_0, s_1, \ldots \) are states; \( s \) is behavior. \( (\forall v: v'/v; \ldots /v') \) denotes substitution for all variables \( v, v', \ldots \)

3. Modeling and analysis of workflows based on temporal logic

3.1 Basic concepts

A basic process modeling language [8] based on a standard process definition notation, which is proposed by workflow management coalition (WfMC) [9], can be used to represent the components of a workflow in a simple
and direct way. In this language, processes are modeled using two types of objects: node and transition. A node is classified into two subclasses: task and choice/merge coordinator. The task, graphically represented by a rectangle, represents the work to be done. It can be used to build implicitly sequence, fork, and synchronous structures. The task is further classified into four types: activity, sub-process, block, and null task, which are necessary for the process modeling. However, for simplicity, all kinds of tasks are treated only as activities when we modeling workflow based on temporal logic. The choice/merge coordinator, graphically represented by a circle, is used to build explicitly choice and merge structures. A transition linking two nodes in a graph is graphically represented by a directed edge and used to specify the execution order and flow between its tail and head nodes. In the later discussion, for being comprehended easily, we will firstly using the activity diagram to show the model of a workflow process, then we will present the temporal logic based model of the same workflow process.

3.2 Modeling workflow process using Temporal Logic of Actions

To modeling workflow process using temporal logic, we must describe the semantic of workflow process using temporal logic. We use Temporal Logic of Actions as our language to describe the semantic of workflow process. Firstly, we use the behavior (an infinite sequence of states) to represent a running of a workflow process. The states in the behavior describe the steps in running of a workflow. In our TLA, we define “Start, End, T1, T2,...” as constants to present the begin, finish, and the activities which be executed in workflow respectively. In order to describe the running of a workflow, we define two set based variables in TLA:

**Definition 1:** P is a set based variable, which the elements in the set P represent the tasks which have been finished in a running of the workflow.

**Definition 2:** S is a set based variable, which the elements in the set S represent the tasks, which are running in the workflow.

Now according to the six workflow control structures show in Figure 1 [10], we give the corresponding description in TLA:

**Figure 1. workflow control structure**

- **Sequence:** Tasks are executed in order under a single thread of execution, which means that succeeding task cannot start until the preceding task is completed. A sequence control structure is described as an action of TLA. The action is denoted by SE:

  **Definition 3:** SE (T_i, T_j) ::= \( \land T_i \in S \)
  \[\land P' = P \cup \{T_i\}\]
  \[\land S' = S \cup \{T_j\} - \{T_i\}\]
  Where \( T_i \) is the preceding task and \( T_j \) is the succeeding task.

- **AND-Split:** A single thread of control splits into two or more threads, which are executed in parallel within the workflow, allowing multiple tasks to be executed simultaneously. An AND-Split control structure is described as an action of TLA. The action is denoted by AS:

  **Definition 4:** AS (T_i, T_{i1}, T_{i2}, ..., T_{in}) ::= \( \land T_i \in S \)
  \[\land P' = P \cup \{T_i\}\]
  \[\land S' = S \cup \{T_{i1}, T_{i2}, ..., T_{in}\} - \{T_i\}\]
  Where \( T_i \) is the task which split and \( T_{i1}, T_{i2}, ..., T_{in} \) are tasks after splitting, \( n \) is the number of splitting.

- **AND-Join:** Two or more parallel executing tasks converge into a single common thread of control. An AND-Join control structure is described as an action of TLA. The action is denoted by AJ:

  **Definition 5:** AJ (T_{i1}, T_{i2}, ..., T_{in}, T_i) ::= \( \land \{T_{i1}, T_{i2}, ..., T_{in}\} \subseteq S \)
  \[\land P' = P \cup \{T_{i1}, T_{i2}, ..., T_{in}\}\]
  \[\land S' = S \cup \{T_i\} - \{T_{i1}, T_{i2}, ..., T_{in}\}\]
Where $T_i$ is the task which is converged into and $T_{i1}$, $T_{i2}$, ... , $T_{in}$ is tasks ready for converging, $n$ is the number of converging.

- **OR-Split**: A single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches. An OR-Split control structure is described as an action of TLA. The action is denoted by $OS$:

  **Definition 6**: $OS(T_i, T_{i1}, T_{i2}, ... , T_{in}) := \vee (OS_1 \wedge \neg OS_2 \wedge \neg OS_3 \wedge ... \wedge \neg OS_n)$ \[ \vee (\neg OS_1 \wedge OS_2 \wedge \neg OS_3 \wedge ... \wedge \neg OS_n) \]

Where $OS_k := \wedge T_i \in S$ \[ \wedge P' = P \cup \{T_i\} \]
\[ \wedge S' = S \cup \{T_{i1}\} - \{T_i\} \]
\[ k = 1, 2, ..., n \]

$OS_k(T_i, T_{i1})$ means the decision is made to choose the branch $T_{i1}$. Where $T_i$ is the task which split and $T_{i1}$, $T_{i2}$, ... , $T_{in}$ is tasks after splitting, $n$ is the number of splitting.

- **OR-Join**: Two or more parallel alternative workflow branches converge into a single common task as the next step within the workflow. No synchronization is required because of no parallel task execution. An OR-Join control structure is described as an action of TLA. The action is denoted by $OJ$:

  **Definition 7**: $OJ(T_{i1}, T_{i2}, ... , T_{in}, T_i) := \vee (OJ_{i1} \wedge \neg OJ_{i2} \wedge \neg OJ_{i3} \wedge ... \wedge \neg OJ_{in})$ \[ \vee (\neg OJ_{i1} \wedge OJ_{i2} \wedge \neg OJ_{i3} \wedge ... \wedge \neg OJ_{in}) \]

Where $OJ_k := \wedge T_{ik} \in S$ \[ \wedge P' = P \cup \{T_{ik}\} \]
\[ \wedge S' = S \cup \{T_i\} - \{T_{ik}\} \]
\[ k = 1, 2, ..., n \]

Where $OJ_k(T_{ik}, T_i)$ means that it is the task $T_{ik}$ which converge into $T_i$.

- **LOOP**: A workflow cycle involves the repetitive execution of one (or more) workflow task until a condition is met. A LOOP control structure is described as an action of TLA. The action is denoted by $LP$:

  **Definition 8**: $LP(T_i) := \wedge T_i \in S$ \[ \wedge P' = P \]
\[ \wedge S' = S \]

To definition a workflow process model, besides the six control structures, we need to describe two additional semantic which represent the initial state of a workflow model running and the state that the workflow model has run to the end:

**Definition 9**: $Init := \wedge P = \Phi$ \[ \wedge S = \{Start\} \]
\[ \wedge P' = \Phi \cup \{End\} \]
\[ \wedge S' = S \]

Where $T_f$ is the start node of the workflow.

**Definition 10**: $Fini := \wedge T_i \in P$ \[ \wedge S = \{Start\} \]
\[ \wedge P' = \Phi \cup \{End\} \]
\[ \wedge S' = S \]

Where $T_f$ is the end node of the workflow.

Now we can define the workflow process model as flow: A workflow process model is described as a formula of TLA. The formula is denoted by $W$:

**Definition 11**:

$W := Init \wedge Finiti \wedge SF[<P, S>](A)$

Where $Init$ and $Finiti$ represent the start and the end of the workflow $W$ repeatedly:

$A := (\vee SE_i) \vee (\vee AS_i) \vee (\vee OSi) \vee (\vee OR_i) \vee (\vee LP_i)$

in formula $\vee SE_i$, $g$ is the number of sequence control structures in $W$, $SE1, SE2, ..., SEg$ are the actions which describe the corresponding sequence control structures, $SE0 = \Phi$. Other formulas in $A$ are explained in the similar way. Now we explain the physical meaning of the definition 11. In a behavior $\sigma$ of TLA, formula $W$ is true means that the first state in the $\sigma$ is the initial of the workflow running, every transformation of the states in $\sigma$ represents one of the six control structures in $W$ is executed, and the states in the $\sigma$ reach to the finish state of the workflow and remain unchanged. That is, a behavior $\sigma$ which makes the formula $W$ be true describes a practical running of the workflow.

### 3.3 Analysis of workflow process model using Temporal Logic of Actions

A property of workflow process model that based on TLA could be expressed by a TLA formula $F$. The assertion “a workflow model $W$ has property $F$” is expressed in TLA by the validity of the formula $W \Rightarrow F$, which asserts that every behavior satisfying $W$ satisfies $F$. So analysis of workflow process model based on TLA is converted to proof some formula true or false. We consider two popular classes of properties:

- **Invariance property**: an invariance property is expressed by a TLA formula $\square P$, where $P$ is a predicate. Examples of invariance properties include $P$ asserts that if the process has terminated, $P$ asserts...
that the process is not deadlocked and so on. Invariance properties could be proved with proof rule 3 of TLA.

- The second classes of properties we consider are inevitability properties—ones asserting that something eventually happens; an inevitability property is expressed by a TLA formula $\diamond P$, where $P$ is a predicate. “The process eventually terminates” and “the resources that a task of the workflow requires will be assigned eventually” are the typical inevitability properties and their expressions in temporal logic are $\diamond P$. Although inevitability properties are expressed by a variety of temporal formulas, their proofs can always be reduced to the proof of “leads-to” formulas, which are defined in section 2. Leads-to formulas could be proved with proof rule 5 of TLA.

4. Case study

In this section we use a case study to illustrate the process of modeling and analysis of workflow model using Temporal Logic of Actions. Firstly we give a simple deadlock freedom workflow process (its activity diagram is shown in Figure 2) for modeling and analysis. Then we change the process to one including deadlock (its activity diagram is shown in Figure 3). Based on the new process, we also modeling it and proof it does have a deadlock using TLA.

![Figure 2. deadlock freedom workflow process W1](image)

**Figure 2. deadlock freedom workflow process W1**

![Figure 3. workflow process W2 including deadlock](image)

**Figure 3. workflow process W2 including deadlock**

Based on the semantic of workflow process in terms of TLA which present in section 3, process shown in Figure 2 could be modeled using TLA as follow:

$$
W1 := \text{Init}_{w1} \land \text{Fin}_{w1} \land \Box [A]_{\Phi, S, S} \land SF_{\Phi, S, S}(A)
$$

Where $\text{Init}_{w1} := \land P = \Phi$

$\land P = \{\text{Start}\}$

$\land S' = T_1$

$\text{Fin}_{w1} := \land T_6 \in P$

$\land S = \{\text{End}\}$

$\land P' = P \cup \{\text{End}\}$

$\land S' = S$

$$
A := OS \lor SE1 \lor SE2 \lor OJ
$$

$OS := (OS_1 \lor \neg OS_2) \lor (\neg OS_1 \lor OS_2)$

$OS_1 := (T_1 \in S) \land (P' = P \cup \{T_1\}) \land (S' = S \cup \{T_2\} - \{T_1\})$

$OS_2 := (T_1 \in S) \land (P' = P \cup \{T_1\}) \land (S' = S \cup \{T_3\} - \{T_1\})$

$OJ := (\neg OS_1 \lor \neg OS_2) \lor (\neg OS_1 \lor \neg OS_2)$

$OJ_1 := (T_2 \in S) \land (P' = P \cup \{T_2\}) \land (S' = S \cup \{T_6\} - \{T_2\})$

$OJ_2 := (T_5 \in S) \land (P' = P \cup \{T_5\}) \land (S' = S \cup \{T_6\} - \{T_5\})$

$SE1 := (T_2 \in S) \land (P' = P \cup \{T_2\}) \land (S' = S \cup \{T_6\} - \{T_2\})$

$SE2 := (T_3 \in S) \land (P' = P \cup \{T_3\}) \land (S' = S \cup \{T_3\} - \{T_3\})$

After modeling the process, we can analyze the properties of the model above. We choose structure conflicts as the property we interest in. In literature [8], deadlock and lack of synchronization are defined as two kinds of structure conflicts in workflow model. Deadlock is defined as a structure conflicts led by an OR-Split eventually join as an AND-Join. A deadlock leads to an AND-Join and its succeeding tasks cannot be executed forever. Obviously, the process above has no deadlock in it. But we can proof it by reasoning in TLA. Here we define a formula to express there is no deadlock in the process:

$$
\Box (\text{Enabled } N) \tag{1}
$$

Where $N := A \lor \text{Fin}_{w1}$, this formula states that there are activities could be executed all the time during the running of the workflow $W1$ unless $W1$ runs to the end, that is there is no deadlock in $W1$.

Want to prove (1), proof 3 tells us that we must prove

$$
\text{Init}_{w1} \Rightarrow \text{Enabled } N \tag{2}
$$

$$
(\text{Enabled } N) \land W1 \Rightarrow \text{Enabled } N' \tag{3}
$$

$$
\text{Init}_{w1} \Rightarrow \text{Enabled } OS, \text{ and Enabled } OS \Rightarrow \text{Enabled } N, \tag{4}
$$

(3) could be decomposed by the definition of $N$:

$$
\text{Enabled } SE1 \land W1 \Rightarrow \text{Enabled } N' \tag{4}
$$

$$
\text{Enabled } SE2 \land W1 \Rightarrow \text{Enabled } N' \tag{5}
$$

$$
\text{Enabled } OS \land W1 \Rightarrow \text{Enabled } N' \tag{6}
$$

$$
\text{Enabled } OJ \land W1 \Rightarrow \text{Enabled } N' \tag{7}
$$

$$
\text{Enabled } \text{Fin}_{w1} \land W1 \Rightarrow \text{Enabled } N' \tag{8}
$$

We prove (8) firstly, Enabled $\text{Fin}_{w1} \land W1 \Rightarrow \text{Enabled } \text{Fin}_{w1} \Rightarrow \text{Enabled } N'$.

Now we prove (4), Enabled $SE1 \land W1 \Rightarrow OS$

$$
\Rightarrow \exists t \in \sigma , \text{ make (Enabled } SE1 \land \neg \text{ Enabled } SE2) \lor (\neg \text{Enabled } SE1 \land \neg \text{Enabled } SE2) \text{ be true,}
$$

$$
\Rightarrow \exists t' \in \sigma , \text{ make } ((\{T_4\} \subseteq t[S]) \land \neg (\{T_3\} \subseteq t[S])) \lor
$$

$$
(\neg \{T_4\} \subseteq t[S]) \land (\{T_3\} \subseteq t[S]) \text{ be true,}
$$

$$
\Rightarrow \exists t' \in \sigma , \text{ make (Enabled } OJ) \text{ be true,}
$$

$$
\Rightarrow \text{Enabled } A' \Rightarrow \text{Enabled } N'. \text{ And (5) could be proved in the similar way.}
$$

(6) and (7) could be proved as follow:

Enabled $OS \land W1$

$$
\Rightarrow \exists t \in \sigma , \text{ make } ((\{T_2\} \subseteq t'[S]) \land \neg (\{T_3\} \subseteq t'[S])) \lor
$$

$$
(\neg \{T_2\} \subseteq t'[S]) \land (\{T_3\} \subseteq t'[S]) \text{ be true,}
$$

where $P$ is a predicate. “The process eventually terminates” and “the resources that a task of the workflow requires will be assigned eventually” are the typical inevitability properties and their expressions in temporal logic are $\diamond P$. Although inevitability properties are expressed by a variety of temporal formulas, their proofs can always be reduced to the proof of “leads-to” formulas, which are defined in section 2. Leads-to formulas could be proved with proof rule 5 of TLA.
∃ t’ ∈ σ, make (Enabled SE1 ∧ ¬ Enabled SE2) ∨ (¬ Enabled SE1 ∧ Enabled SE2) be true,
⇒ Enabled A’ ⇒ Enabled N’. And (7) could be proved in the similar way.

So far we have proved (2) and (3). By the definition of $W_1$, we have $W_1 ⇒ Init_{w1} ∧ □[A]_{P, S}$, considering (2), we have

$W_1 ⇒ Enabled N ∧ □[A]_{P, S}$  \hspace{1cm} (9)

Applying proof rule 3 to (3) and (9), we have $W_1 ⇒ □ (Enabled N)$

that is there is no deadlock in $W_1$.

Now we change the process above by substituting the OR-Join structure by an AND-Join structure. The TLA model of new process $W_2$ is similar with $W_1$. We can get $W_2$ by substituting the definition $AJ$ of $W_1$ by the definition $AJ$:

$AJ (T_4, T_5, \ldots, T_6) ::= \wedge \{ T_4, T_5 \} \subseteq S$

$∧ P'^* = P \cup \{ T_4, T_5 \}$

$∧ S'^* = S \cup \{ T_6 \} \setminus \{ T_4, T_5 \}$

The process $W_2$ includes a deadlock in it. We can proof it by reasoning in TLA. Here we express there is a deadlock in the process formally as a TLA formula:

$□ \neg \{ \{ T_4, T_5 \} \subseteq S \}$  \hspace{1cm} (10)

(10) asserts that during the running of the $W_2$, the AND-Join of $T_4$ and $T_5$ could be executed. So (10) represent there is a deadlock in the process. Now we use apagoge to prove (10). Supposing that $∃ t’ ∈ σ$ make $\{ T_4, T_5 \} \subseteq S'$, where behavior $σ$ is a running of the workflow $W_2$. By the definition of $W_1$, we have $∃ t’ ∈ σ$, make $\{ T_2, T_3 \} \subseteq P^*$  \hspace{1cm} (11)

by (11) and the definition of $W_1$, we have $∃ t ∈ σ$, make $(OS_1 \lor OS_2)$ be true  \hspace{1cm} (12)

that is, $∃ t ∈ σ$ make formula $¬W_2$ be true. So $σ$ is not a running of the workflow $W_2$, this conflicts with our assumption. Now we have proved (10) which asserts that $W_2$ includes a deadlock.

5. Conclusions and future work

In this paper, we develop a framework for modeling and analyzing workflows based on Temporal Logic of Actions. Rather than collecting all the information need for a complicated business workflow, the power of this modeling method is analysis of workflow model by reasoning in TLA. There are two significant problems remaining to be studied. The one is the mechanical verification in TLA. Because it is a simple logic, TLA is ideally suited for mechanization. The mechanical verification in TLA will help us analyze model based on TLA more efficiently. The other problem is to design a model satisfied some particular properties. In TLA, there is no distinction between a model and a property. We can just as well consider a model to be a property that we want another model to satisfy.

References