Computational Complexity of Web Service Composition
Based on Behavioral Descriptions

Hyunyoung Kil  Wonhong Nam  Dongwon Lee*
The Pennsylvania State University, University Park, PA 16802, USA
E-mail: {hkil, wnam, dongwon}@psu.edu

Abstract

The Web Service Composition (WSC) problem on behavioral descriptions deals with the automatic construction of a coordinator web service to control a set of web services to reach the goal states. As such, WSC is one of the fundamental techniques to enable the Service Oriented Architecture on the Web. Despite its importance and implications, however, very few studies exist on the computational complexities of the WSC problem. In this paper, we present two novel theoretical findings on WSC problems: (1) Solving the WSC problem with “complete” information is EXP-hard, and (2) Solving the WSC problem with “incomplete” information is 2-EXP-hard. These findings imply that more efforts to devise efficient approximate solutions to the WSC problem be needed.

1. Introduction

Web services [7] are software systems designed to support machine to machine interoperation over the Internet. Given a set of web services and a user request, the Web Service Composition (WSC) problem is to find a composition of web services satisfying the request. In this problem, a coordinator web service controls the involved web services to achieve the given goal, while the involved web services do not know the composition. The focus of this paper is, given a set \( W \) of (behavioral descriptions of) web services and a reachability goal \( G \), to synthesize a coordinator web service, \( c \), that controls \( W \) to satisfy \( G \).

There are abundant researches on the WSC problem (e.g., [6, 1, 3, 4]). However, to the best of our knowledge, only a few (e.g., [6, 1]) underlie realistic models (i.e. with incomplete information). Moreover, few studies exist, which have investigated the computational complexity (i.e., lower bound) of the WSC problem. Therefore, in this paper, we present two complexity results of the WSC problem.

2. Web Service Composition

As an example, consider that clients want to make reservations for both of a flight ticket and a hotel room for a particular destination and a period. However, only an airline reservation (AR) web service and a hotel reservation (HR) web service have been built separately. Clearly, we want to combine these two web services rather than implementing a new one. One way to combine them is to construct a coordinator web service (Travel agency system) which communicates with AR and HR to book up a flight ticket and a hotel room together. Fig 1 illustrates this example.

A web service \( w \) is a tuple \( (X, X^I, X^O, Init, T) \) where

- \( X \) is a finite set of variables that \( w \) controls; a state \( s \) of \( w \) is a valuation for every variable in \( X \), and we denote a set of all the states as \( S \).
- \( X^I \) is a finite set of input variables which \( w \) reads from its environment; \( X \cap X^I = \emptyset \), and every variable \( x \in X \cup X^I \) has a finite domain (e.g. Boolean, bounded integers, or enumerated types). A state \( s \) in inputs is a valuation for every variable in \( X^I \), and we denote a set of all the input states as \( S^I \).
- \( X^O \subseteq X \) is a finite set of output variables that its environment can read.
- \( Init(X) \) is an initial predicate over \( X \); \( Init(s) = true \) iff \( s \) is an initial state.

*Partially supported by IBM and Microsoft gifts.
• $T(X, X', X')$ is a transition predicate over $X \cup X' \cup X'$. For a set $X$ of variables, we denote the set of primed variables of $X$ as $X' = \{x' \mid x \in X\}$, which represents a set of variables encoding the successor states. $T(s, in, s')$ is true iff $s'$ can be a next state when the input $in \in S'$ is received at the state $s$.

Given a state $s$ over $X$ and a variable $x \in X$, $s(x)$ is the value of $x$ in $s$. For a state $s$ over $X$, let $s[Y]$ where $Y \subseteq X$ denote the valuation over $Y$ obtained by restricting $s$ to $Y$. Note that the process model for most web services described in Semantic Web languages (e.g., OWL-S or WSBPEL) can be easily transformed into our representation above without any information loss.

In the WSC problem, given a set of available web services, $W$, every service in $W$ communicates only with their coordinator but not with each other. Based on this assumption, we define a non-deterministic transition relation, in this case it is a tuple $G = \langle X, X', X^O, Init, T \rangle$ where

- $X = \bigcup_i X_i, X' = \bigcup_i X'_i, X^O = \bigcup_i X^O_i$.
- $Init(X) = \bigwedge_i Init_i, T(X, X', X') = \bigwedge_i T_i$.

Since a coordinator web service is also a web service, it is a tuple $c = (X_c, X'_c, X^O_c, Init_c, T_c)$.

Although $T_c$ can define a non-deterministic transition relation, in this problem we want only a deterministic transition relation for $c$; i.e., for every $s$ and $in$, there exists only one $s'$ such that $T_c(s, in, s') = true$. Then, given a set $W = \langle X, X', X^O, Init, T \rangle$ of web services and a coordinator $c = (X_c, X'_c, X^O_c, Init_c, T_c)$ (in what follows, we assume that $X'^c = X^O_c$ and $X^O_c = X'_c$), we can define an execution tree, denoted by $W[c]$, which represents the composition of $W$ and $c$ as follows:

- Each node in $W[c]$ is in $S \times S_c$. The root node is $(s, s_c)$ such that $Init(s) = true$ and $Init_c(s_c) = true$.
- For each node $(s, s_c)$, it has a set of child nodes, $\{(s', s'_c) \mid T(s, in, s') = true, in = s_c[X'_c], T_c(s_c, in_c, s'_c) = true, in_c = s'_c[X^O_c]\}$. Intuitively, the web services $W$, by receiving the input $in$ from the current state $s_c$ of the coordinator, proceeds from $s$ to the next state $s'$, and then the coordinator, by receiving the input $in_c$ from the new state $s'_c$ of the web services, proceeds from $s_c$ to the next state $s'_c$.

A goal $G \subseteq S$ is a set of states where we want to reach. Given a set $W$ of web services, a coordinator $c$, and a goal $G$, we define $W[c] \models G$ if for every path $(s^0, s^1, s^2, \ldots)$ in the execution tree $W[c]$, there exists $i \geq 0$ such that $s^i \in G$; namely, every path from the initial node $(s^0, s^0)$ reaches a goal state eventually. Finally, a web service composition (WSC) problem that we focus on in this paper is, given a set $W$ of web services and a goal $G$, to construct a coordinator web service $c$ such that $W[c] \models G$.

The problem size is the number of Boolean variables when we encode all variables in $X$ into Boolean variables.

To study the computational complexity, we define two WSC problems as follows:

- **WSC with complete information**: a special case of WSC problems where $W = \langle X, X', X^O, Init, T \rangle$ such that $X = X^O$; $W$ has no internal variable.
- **WSC with incomplete information**: a general WSC problem where there is no restriction for $X^O$. That is, a coordinator can read only the output variables for $W$.

### 3. Alternating Turing Machine

#### 3.1 Definition

An alternating Turing machine (ATM) [5] is a tuple $A = \langle Q, \Sigma, q_0, \delta, l \rangle$ where

- $Q$ is a finite set of states, $\Sigma$ is a finite tape alphabet, and $q_0 \in Q$ is the initial state.
- $\delta : Q \times \Sigma \rightarrow 2^{Q \times \Sigma \times \{\#\} \times \{L, N, R\}}$ is a transition function where $\{L, N, R\}$ represents the R/W head movement (i.e., it moves left, stays, or right).
- $l : Q \rightarrow \{\forall, \exists, \text{accept}\}$ is a labeling function for states.

A configuration of an ATM $A(Q, \Sigma, q_0, \sigma, l)$ is a tuple $(q, \sigma, \sigma')$ where $q \in Q$ is the current state, $\sigma \in \Sigma^*$ is the tape contents left of the R/W head with the rightmost symbol under the R/W head, and $\sigma' \in \Sigma^*$ is the tape contents strictly right of the R/W head. Given an ATM $A$ with an input string $a\sigma$, the initial configuration is $(q_0, a, \sigma)$.

Given an ATM $A$ and its input string $\sigma$, to see if $A$ accepts $\sigma$ (i.e., $\sigma \in L(A)$), we define $n$-accepting for configurations in its computation tree by a bottom-up manner:

- $(q, \sigma, \sigma')$ is 0-accepting if $l(q) = \text{accept}$.
- $(q, \sigma, \sigma')$ such that $l(q) = \forall$ is $n$-accepting if all the successor nodes are $m$-accepting for some $m < n$ and $\max(m) = n-1$.
- $(q, \sigma, \sigma')$ such that $l(q) = \exists$ is $n$-accepting if at least one of its child nodes is $m$-accepting for some $m < n$ and $\min(m) = n-1$.

Finally, $A$ accepts $\sigma$ if the initial configuration is $n$-accepting for some $n \geq 0$. For the detail of ATMs, see [5].

#### 3.2 Computational complexity

In this paper, we consider the following complexity classes [5]. $\text{DTIME}(f)$ is a time consumption complexity on deterministic Turing machines, and $\text{DSPACE}(f)$ is a space consumption complexity on deterministic Turing machines. Similarly, we have $\text{ATIME}(f)$ and $\text{ASPACE}(f)$ as time and space consumption complexities on alternating Turing machines (ATMs), respectively.
Then, the class APSPACE (resp. AEXPSPACE) is the set of decision problems that can be solved by an ATM using a polynomial (resp. exponential) amount of memory, respectively. Chandra et al. [2] have proved interesting connections between time complexities on deterministic Turing machines and space complexities on ATMs:

**Theorem 1.** \( \text{EXP} = \text{APSPACE} \), and \( \text{2-EXP} = \text{AEXPSPACE} \).  

In Section 4, we will show the space complexities for the WSC problems. Then, by the connections above, the space complexities imply the time complexities of the problems.

### 4. Lower Bounds

In this section, we study the computational complexities (lower bounds) for two WSC problems we defined in Sec 2.

#### 4.1 WSC with complete information

**Theorem 2.** The WSC problem with complete information is EXP-hard (i.e. APSPACE-hard).

The proof is to simulate an ATM with a polynomial tape length. That is, for any ATM \( A \) and an input string \( \sigma \), we can construct a WSC problem in polynomial time such that \( A \) accepts \( \sigma \) iff there exists a coordinator to satisfy a goal.

**Lemma 1.** Given an ATM \( A = (Q, \Sigma, q_0, \delta, \lambda) \) with polynomial space bound \( p(n) \) and an input string \( \sigma = a_1 \cdots a_n \), we can construct a WSC problem instance.

**Proof.** We can construct a set \( W(X, X', X^O, Init, T) \) of web services and a goal \( G \) which have a polynomial size in the size of the description of \( A \) and \( \sigma \) as follows. The set \( X \) of variables includes the following variables:

- \( \text{state} \) represents the current state of \( A \); so, it has the domain. \( \{q \mid q \in Q\} \).
- For \( 1 \leq i \leq p(n) \), \( cl_i \) has the contents of the \( i \)-th tape cell; its domain is \( \Sigma \cup \{\#\} \).
- \( hd \) describes the R/W head position; its domain is \( \{1, \cdots, p(n)+1\} \).
- \( label \) represents the label of the current state; it has the domain, \( \{\forall, \exists, \text{accept}\} \).

The set \( X' \) is \( \{\text{input}\} \) where the domain of \( \text{input} \) is \( \{A_{q,i,a} \mid q \in Q, l(q) = \forall, 0 \leq i \leq p(n), a \in \Sigma \cup \{E_{q,i,a,j} \mid q \in Q, l(q) = \exists, 0 \leq i \leq |\delta(a, q)|\} \}. \) The set \( X^O \) equals \( X \) since this problem is the complete information problem. As the initial configuration of \( A \), the initial state predicate \( \text{Init}(X) \) is \( (\text{state}=q_0) \land \bigwedge_{1 \leq i \leq n}(cl_i = a_i) \land \bigwedge_{n < i \leq p(n)}(cl_i = \#) \land (hd = 1) \land (label = l(q_0)) \). Note that the input string is \( \sigma = a_1 \cdots a_n \). The transition predicate \( T(X, X', X^O) \) is \( (hd = p(n)+1) \rightarrow T_V \land \bigwedge_{\text{label} = \forall} \rightarrow T_\exists \land (\text{label} = \exists) \rightarrow T_A \) with the following subformulas:

- \( T_V \equiv (\text{state}'=\text{state}) \land (hd'=hd) \land (label'=label) \)
- \( T_V \equiv \bigwedge_{q \in Q, 1 \leq i \leq p(n), a \in \Sigma}(((\text{state}=q) \land (hd=i)) \land (cl_i = a) \land (input=A_{q,i,a})) \rightarrow \bigvee_{1 \leq j \leq k}(\text{state}'=q_j) \land (cl_j = a_j) \land \bigwedge_{m \neq i}(cl_m = cl_m) \land (hd'=hd+\Delta) \land (label' = l(q_j))) \)
- \( T_\exists \equiv \bigwedge_{q \in Q, 1 \leq i \leq p(n), a \in \Sigma}(1 \leq j \leq k) (((\text{state}=q) \land (hd = i)) \land (cl_i = a) \land (input=A_{q,i,a})) \rightarrow (((\text{state}'=q_j) \land (cl_j = a_j) \land \bigwedge_{m \neq i}(cl_m = cl_m)) \land (hd'=hd+\Delta) \land (label' = l(q_j))) \)

where \( (q_j, a_j, m_j) \) is obtained from \( \delta(q, a) = \{(q_1, a_1, m_1), \cdots, (q_k, a_k, m_k)\} \), and \( \Delta = -1 \) if \( m_j = \lambda \), \( \Delta = 0 \) if \( m_j = \lambda \), and \( \Delta = 1 \) if \( m_j = \lambda \). Note that the value for the variable \( input \) is provided by a coordinator \( c \). Finally, we have a goal, \( G = \{s \in S \mid s(\text{label}) = \text{accept}\} \).

If the ATM \( A \) violates the space bound, \( hd \) has the value \( p(n)+1 \), and after this point we cannot reach goal states since \( W \) stays the same state forever by \( T_V \). (Q.E.D)

**Lemma 2.** If \( \sigma \in L(A) \), then there exists a coordinator \( c = (X_c, X'_c, X^O_c, \text{Init}_c, T_c) \) such that \( W(c \mid \sigma) = G \).

**Proof.** \( \sigma \in L(A) \) means that the initial configuration of \( A \) with respect to \( \sigma \) is \( m \)-accepting. When \( A \) accepts \( \sigma \), we can define an accepting computation tree \( ACT_{(A, \sigma)} \) of \( A \) with respect to \( \sigma \) from its computation tree \( \mathcal{T} \) as follows:

- For each configuration \( cf = (q, \sigma_1, \sigma_2) \in \mathcal{T} \) such that \( l(q) = \forall \), all the successor configuration are also included in \( ACT_{(A, \sigma)} \). Note that if \( cf \) is \( m \)-accepting, each successor is at most \( (m-1) \)-accepting.
- For each \( cf = (q, \sigma_1, \sigma_2) \in \mathcal{T} \) such that \( l(q) = \exists \) and \( cf \) is \( m \)-accepting, only one successor configuration \( cf' \) which is \( (m-1) \)-accepting is included in \( ACT_{(A, \sigma)} \).

When \( A \) and \( \sigma \) are clear from the context, we drop the subscript \((A, \sigma)\) and write \( ACT \). Let \( \sigma[i] \) be the \( i \)-th symbol of the string \( \sigma \). Now, we show that there exists a coordinator \( c \) such that for every path \( (s^0, s^0)((s^1, s^1) \cdots \cdot \cdot \cdot) \) in the execution tree \( W(c \mid \sigma) \), there exists \( s^0 \in G \). The coordinator to be constructed is \( c = (X_c, X'_c, X^O_c, \text{Init}_c, T_c) \) where \( X_c = \{\text{input}\} \), \( X'_c = X \), and \( X^O_c = \{\text{input}\} \).

Also, we can define \( T_c \) with a conjunction of two cases: \( \forall \)-state and \( \exists \)-state. That is, if \( l(q) = \forall \), the transition
predicates is \( \bigwedge_{q \in Q, 1 \leq i \leq p(n), a \in \Sigma} (((state = q) \land (label = \forall) \land (hd = i) \land (cl_i = a)) \rightarrow (input' = A(q,i,a))) \). Otherwise, \( \bigwedge_{q \in Q, 1 \leq i \leq p(n), a \in \Sigma} (((state=q) \land (label=\exists) \land (hd = i) \land (cl_i=a)) \rightarrow (input'=E(q,i,a,j))) \) where \( j \) is the index of the transition by which the ATM proceeds from the corresponding \( \exists \)-configuration to the next in \( ACT(A,\gamma) \). Similarly with \( T_c \), we can define the initial predicate \( Init_c \) as \(((l(q_0) = \forall) \land (input = A(q_0,0,0)) \land (l(q_0) = \exists) \rightarrow (input = E(q_0,0,0))) \) where \( a_1 \) is the first symbol of the input string \( \sigma \) and \( j \) is obtained as the above.

Then, \( ACT(A,\sigma) \) is mapped into an execution tree \( W|c \).

For this mapping, we have two mapping functions, \( \alpha \) and \( \beta \); \( \alpha \) maps a configuration \( cf \) of \( ACT \) to a state \( s \) of web services \( W \), and \( \beta \) maps \( cf \) to a state \( s_c \) of the coordinator \( c \). First, for each \( cf = (q, \sigma_1, \sigma_2) \), we have a corresponding state \( s = \alpha(cf) \) of \( W \) such that

- \( s(state) = q \).
- For \( 1 \leq i \leq |\sigma| \), \( s(cl_i) = \sigma[i] \) where \( \sigma = \sigma_1 \sigma_2 \), and for \( |\sigma| < i \leq p(n) \), \( s(cl_i) = \# \).
- \( s(hd) = |\sigma_1| \).
- \( s(label) = l(q) \).

Next, for each configuration \( cf = (q, \sigma_1, \sigma_2) \), we have a corresponding state \( s_c = \beta(cf) \) of \( c \) such that

- If \( l(q) = \forall \), then \( s_c(input) = A(q,i,a) \) where \( i = |\sigma_1| \) and \( a = \sigma_1[i] \).
- In the case of \( l(q) = \exists \), let \( cf' \) be the only successor of \( cf \) in \( ACT \), which is obtained by a transition \((q_j, a_j, m_j)\) among \( \delta(q,a) = \{(q_1, a_1, m_1), \ldots, (q_k, a_k, m_k)\} \) where \( a = \sigma_1[|\sigma_1|] \).

Now, if \( l(q) = \exists \), \( s_c(input) = E(q,i,a,j) \) where \( i = |\sigma_1| \) and \( a = \sigma_1[|\sigma_1|] \).

According to \( \alpha \) and \( \beta \), we have an execution tree of \( W|c \) where each node is \((\alpha(cf), \beta(cf)) \). Now, we can show by induction, that if \( cf \) in \( ACT \) is \( m \)-accepting, every path from the corresponding node \((\alpha(cf), \beta(cf)) \) in \( W|c \) reaches a goal state eventually. We, however, omit the induction for the sake of space. Finally, since the initial configuration is \( m \)-accepting, every path from the initial node of \( W|c \) reaches a goal state; i.e., \( W|c \models G \). (Q.E.D)

**Lemma 3.** If there exists a coordinator \( c \) such that \( W|c \models G \), then \( \sigma \in L(A) \).

**Proof.** As shown in Sec 2, the fact that there exists a coordinator \( c \) such that \( W|c \models G \) means that every path \((s^0, s^1, s^2) \ldots \) from the initial node in the execution tree \( W|c \) reaches a goal state eventually. Now, we show that an \( ACT \) for \( A \) corresponding to \( W|c \) can be constructed and the initial configuration is \( m \)-accepting.

We denote as \( ST \), a finite subtree of \( W|c \) which includes, for every path \((s^0, s^1, s^2) \ldots \) of \( W|c \), its prefix ending at a goal state (i.e. \((s^0, s^1, s^2) \ldots (s^k, s^k)\) such that \( s^k \in G \)). We construct an \( ACT \) for the ATM \( A \) from the subtree \( ST \). For the mapping, we have a mapping function \( \gamma \) which maps a state \( s \) of web services \( W \) to a configuration \( cf \) of \( A \). For each state \( s \) such that \( s(state) = q \), \( s(cl_i) = b_i \), where \( 1 \leq i \leq p(n) \), \( s(hd) = i \) and \( s(label) = l(q) \), we have a corresponding configuration \( cf = (q, \sigma_1, \sigma_2) \) such that \( \sigma_1 = b_1 \ldots b_i \) and \( \sigma_2 = b_{i+1} \ldots b_{k-1} \) where \( k \) is the index of the first appearance of \( \# \).

Now, we claim that if among every path from a node \((s, s_c)\) to a goal in \( ST \), the length of the longest one is \( m \), the corresponding configuration \( \gamma(s) \) is \( m \)-accepting. We can show by induction on \( m \) that our claim is correct, but we omit it for space. Finally, since the initial node \((s, s_c)\) of \( ST \) has \( m \) (for some \( m \geq 0 \)) as the length of the longest path to a goal, the corresponding configuration \( \gamma(s) \), namely the initial configuration of \( A \), is \( m \)-accepting. (Q.E.D)

### 4.2 WSC with incomplete information

**Theorem 3.** The WSC problem with incomplete information is 2-EXP-hard (i.e. AEXPSPACE-hard).

The proof is to simulate an ATM with exponential tape length. As Theorem 2, we prove it by the following lemmas.

**Lemma 4.** Given an ATM \( A = (Q, \Sigma, q_0, \delta, \delta) \) with exponential space bound \( e(n) \) and an input string \( \sigma = a_1 \ldots a_n \), we can construct a WSC problem instance.

**Proof.** An important difference with the complete information problem is that we are not allowed to have a variable for each tape cell since the number of tape cells is exponential. Instead of including an exponential number of variables \( c_i \), we have one variable \( cl \) and its index \( idx \). The trick is to establish that if the index matches the current head position, \( W \) should simulate the ATM \( A \), and to force the above to be satisfied universally for every index \( idx \). Given an ATM \( A \) with \( \sigma \), we construct a set \( (X, X', X^O, Init, T) \) of web services and a goal \( G \) as follows. The set \( X \) of variables includes the following ones:

- \( state \); its domain is \( \{q | q \in Q\} \).
- \( idx \); its domain is \( \{1, \ldots, e(n)\} \).
- \( cl \) represents the contents of the cell of which index is \( idx \); its domain is \( \Sigma \cup \{\#\} \).
- \( hd \); its domain is \( \{1, \ldots, e(n)+1\} \). For \( idx \) and \( hd \), we need only \( \log_2(e(n)+1) \) bits.
- \( label \); it has a domain, \( \{\forall, \exists, accept\} \).
- \( lsb \) represents the symbol written by the head in the last step; it has a domain, \( \Sigma \cup \{\#\} \).

The set \( X' \) is \( \{input\} \) where the domain of \( input \) is \( \{A(q,a) | q \in Q, l(q) = \forall, a \in \Sigma \cup \{E(q,a,\#) | q \in Q, l(q) = \exists, a \in \Sigma, 0 \leq j \leq |\delta(a,q)|\} \}. The set \( X^O \) is \( \{state, cl, \} \). Init(X)
is ($state = q_0$) \& \((\text{idx} \leq |\sigma|) \Leftrightarrow (cl = a_{\text{idx}}) \& ((\text{idx} > |\sigma|) \Leftrightarrow (cl=#)) \& \((\text{hd}=1) \& (\text{label} = l(q_0))\). The transition predicate $T_{X} = T_{X}^{I}$ is $(\text{hd} = e(n)\text{I}) \rightarrow T_{V}$ with $(\text{label} = \forall \rightarrow T_{V}) \& ((\text{label} = \exists) \rightarrow T_{3})$ with the following sub-formulae:

- $T_{V} \equiv (state = state) \& (\text{hd} = \text{hd}) \& (\text{label} = \text{label})$
- $T_{V} \equiv \bigwedge_{q \in Q, a \in \Sigma} (((state = q) \& ((\text{hd} = \text{idx}) \rightarrow (cl = a)) \& (input = A(q,a))) \rightarrow (\text{hd} = \text{idx}) \rightarrow ((state = q_1) \& (cl = a_j) \& \((\text{hd} = \text{hd} + \Delta) \& (\text{label} = l(q_j)) \& (\text{label} = l(q_j)))$)
- $T_{3} \equiv \bigwedge_{q \in Q, a \in \Sigma, 1 \leq j \leq k} (((state = q) \& ((\text{hd} = \text{idx}) \rightarrow (cl = a)) \& (input = E(a,q,j))) \rightarrow (((\text{hd} = \text{idx}) \rightarrow (state = q_j) \& (cl = a_j) \& \((\text{hd} = \text{hd} + \Delta) \& (\text{label} = x)))$)

where $(q_j, a_j, m_j)$ is obtained from $\delta(q, a) = ((q_1, a_1, m_1), \ldots, (q_k, a_k, m_k))$ and $\Delta = -1$ if $m_j = \mathcal{L}$, $\Delta = 0$ if $m_j = \mathcal{N}$ and $\Delta = 1$ if $m_j = \mathcal{R}$. Finally, we have a goal, $G \equiv \{s \in S | s(\text{label}) = \text{accept}\}$.

If the ATM avoids the space bound, the variable $hd$ has the value $e(n) + 1$, and after this point we cannot reach goal states by $T_{V}$.

**Lemma 5.** If $\sigma \in L(A)$, then there exists a coordinator $c$ such that $W || c \models G$.

**Proof.**

Given a suffix $\sigma$ such that $\sigma \in L(A)$, we can construct a coordinator $c = (X_c, X_c^{I}, X_c^{O}, Init_c, T_{c})$ where $X_c = \{input\}, X_c^{I} = \{state, lsb\}$, and $X_c^{O} = \{input\}$. Like the proof of Lemma 2, we can define $T_{c}$ with a conjunction of two cases: $\forall$-state and $\exists$-state. That is, if $l(q) = \forall$, the transition predicate is $\bigwedge_{q \in Q, a \in \Sigma} (((state = q) \& (cl = a)) \rightarrow (input' = A(q,a)))$. Otherwise, $\bigwedge_{q \in Q, a \in \Sigma} (((state = q) \& (cl = a)) \rightarrow (input' = E(a,q,j)))$ where $j$ is the index of the transition by which the ATM proceeds from the corresponding $\exists$-configuration to the next in $ACT_{(A,a)}$. Similarly with $T_{c}$, we can define the initial predicate $Init_c$ as $((l(q_0) = \forall) \rightarrow (input = A(q_0,a_1))) \& ((l(q_0) = \exists) \rightarrow (input = E(q_0,a_1,j)))$ where $a_1$ is the first symbol of the input string $\sigma$ and $j$ is obtained as described above.

Now, we can show that $ACT_{(A,a)}$ is mapped into an execution tree $W || c$ by two mapping functions, $\alpha$ and $\beta$ like Lemma 2. Then, we claim that if $c$ in $ACT$ is $m$-accepting, then for every $1 \leq i \leq e(n)$ every path from the corresponding node $(\alpha(cf, i), \beta(cf))$ reaches a goal state eventually. By using the property that $T$ and $T_{c}$ strictly follow the transition function $\delta$ of $A$, we can prove the claim by induction on $m$. We omit the mapping functions, $\alpha$ and $\beta$, and the induction for the space limit. Finally, since the initial configuration of $ACT$ is $m$-accepting, every path from the initial node of $W || c$ reaches a goal state; that is, $W || c \models G$.

**Lemma 6.** If there exists a coordinator $c$ such that $W || c \models G$, then $\sigma \in L(A)$.

**Proof.** For the finite subtree $ST$ of $W || c$, we construct $ACT_{(A,a)}$. However, unlike Lemma 3, we are not able to construct a configuration directly from a state of $W$ since $W$ does not have all the tape contents, but only $cl$ and $lsb$. Now, our trick is to construct the computation tree by a top-down manner. Even though the initial state of $W$ has only $cl$ and $lsb$, we can construct the initial configuration as $cf = (q_0, a, \sigma')$ where the input string $\sigma = a\sigma'$. Given a predecessor configuration $cf_1 = (q_1, \sigma_1, \sigma'_1)$ and a state $s$ of $W$ such that $s(state) = q, s(cl) = a_1, s(idx) = i, s(hd) = h, s(label) = l(q), s(lsb) = l(a)$, our mapping function $\gamma$ maps $s$ to a configuration $cf_2 = (q, \sigma_2, \sigma'_2)$ where $|\sigma_2| = h$, and $\sigma_2, \sigma_2'$ is copied from $\sigma_1, \sigma'_1$ except $(\sigma_2, \sigma'_2) || (\sigma_1) = a_2$. Now, we claim that if among every path from a node $(s, s_c)$ to a goal in $ST$, the length of the longest one is $m$, the corresponding configuration $\gamma(s)$ is $m$-accepting. By using the property that our $T$ and $T_{c}$ strictly follow the transition function $\delta$ of $A$, we can prove the claim by induction on $m$. We omit the induction. Finally, since the initial node $(s, s_c)$ of $ST$ has $m$ (for some $m \geq 0$) as the length of the longest path to a goal, the initial configuration of $A$ is $m$-accepting.

**5. Conclusion.**

In this paper, we have studied the computational complexity for two WSC problems on behavioral descriptions. The first one is the WSC problem with complete information where a coordinator to be constructed knows the exact state of a given set of web services. The second is the WSC problem with incomplete information where the coordinator knows only the values of output variables of web services. The main findings of this paper is that (1) the WSC problem with complete information is EXP-hard, and (2) the WSC problem with incomplete information is 2-EXP-hard. These findings imply that it is needed to devise sound lower-complexity approximations for the WSC problem.

**References**


