Theoretical Computer Science 412 (2011) 6736–6749

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# On the computational complexity of behavioral description-based web service composition☆

Wonhong Nam [a], Hyunyoung Kil [b,*], Dongwon Lee [c]

[a] Konkuk University, Seoul 143-701, Republic of Korea

[b] Korea Advanced Institute of Science & Technology, Daejeon 305-701, Republic of Korea

[c] The Pennsylvania State University, University Park, PA 16802, USA

ARTICLE INFO

*Article history:*
Received 16 March 2010
Received in revised form 15 March 2011
Accepted 14 April 2011
Communicated by O. Watanabe

*Keywords:*
Web service composition
Computational complexity
Incomplete information
Behavioral description

ABSTRACT

The behavioral description-based Web Service Composition (WSC) problem deals with the automatic construction of a coordinator web service that controls a set of web services to reach the goal states. Despite its importance and implications, very few studies exist on the computational complexities of the WSC problem. In this paper, to address this problem, we present four novel theoretical findings on the WSC problem: (1) solving the composition problem of *deterministic* web services for a restricted case (when the coordinator web service has *complete information* about the states of all web services) is *PSPACE-complete*; (2) solving the composition problem of *deterministic* web services for a general case (when the coordinator web service has *incomplete information* about the states of web services) is *EXPSPACE-complete*; (3) solving the composition problem of *non-deterministic* web services on *complete information* is *EXP-complete* and (4) solving the composition problem of *non-deterministic* web services on *incomplete information* (which is the most general case) is *2-EXP-complete*. These findings suggest that more efforts to devise efficient approximation solutions to the WSC problem is needed.



## 1. Introduction

*Web services* are software systems designed to support machine to machine interoperation over the Internet. Recently, much research [27] has been carried out for the web service standards based on semantic web techniques, significantly improving the flexible and dynamic functionalities of the *Service Oriented Architectures (SOA)*. However, abundant research challenges still remain [13], e.g., automatic web service discovery [3], web service composition [23,22,5], or formal verification of composed web services [19,11].

In general, the *Web Service Composition (WSC)* problem is, given a set of web services and a user request, to find a composition of web services satisfying the request. The web service composition depends on what level of information we use in composition. The *Web Service Description Language (WSDL)*, acknowledged as an essential building block of web service stack, mainly defines the signature of web services, but it does not support the specification of service workflows. When one considers only the signatures of web services (e.g., in WSDL [30]) as an input of the WSC problem, this problem amounts to identifying *only* a sequence of web services to satisfy the request by parameter matching. In this case, users can invoke the sequence of web services computed, but they cannot react exactly to the output values returned from web services in the sequence during runtime. In the composition area, the service behavioral description languages such as

☆ A preliminary version (Kil and Nam, 2008) [15] of this paper was presented at the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2008. This work was supported by the faculty research fund of Konkuk University in 2010.

* Corresponding author. Tel.: +82 10 2640 0507.
*E-mail addresses:* wnam@konkuk.ac.kr (W. Nam), hkil@kaist.ac.kr (H. Kil), dongwon@psu.edu (D. Lee).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.04.020

**Table 1**
The summary of computational complexities for the WSC problem.

| | With complete information | With incomplete information |
|---|---|---|
| Deterministic WSC | PSPACE-complete | EXPSPACE-complete |
| Non-deterministic WSC | EXP-complete | 2-EXP-complete |

*Web Service Business Process Execution Language (WS-BPEL)* [21] have the more prominent status. This kind of languages specify service behaviors and interactions with other services as a sequence of the activities. In addition, *OWL-S* [28], a well-known Semantic web language, is also able to describe the behavioral specification of web services as *Process Model*. In the languages, the behaviors of web services are specified as a state-transition system, or a set of inputs, outputs, preconditions, and effects. The specification formally describes that on receiving a specific input, what values for outputs a web service will return and what state it will proceed to. The composition with this information computes a strategy suggesting the order of web services—for a certain output value, which web service should be invoked in next with a certain input value in order to satisfy a user provided goal. In general, the strategy to guarantee achieving the goal can be represented as a state-transition system called a *coordinator*. The behavioral description-based web service composition makes more useful suggestion that can be executed in runtime. Therefore, given a set of behavioral descriptions of web services, $W$, and a reachability goal, $G$, *Web Service Composition (WSC)* problem that we focus on in this paper is to synthesize a *coordinator web service*, $c$, that controls $W$ to satisfy $G$.

There is abundant research on the WSC problem (e.g., [23,13,22,5,18]). However, only a few (e.g., [23,22,5]) are based on realistic models (i.e., they are the only ones which deal with *incomplete information*) for the behavioral description-based web service composition. Moreover, to the best of our knowledge, no study has investigated the computational complexity of the WSC problem with complete proofs. Our investigation into the complexity can provide valuable insights to precisely understand the WSC problem, to know what is possible, and to identify interesting sub-problems. We first present a simple example, a *travel agency system*, to illustrate this problem, and formally define a web service with a state-transition system and several different problem settings.

The first classification is *deterministic*/*non-deterministic* web services; namely, in a deterministic web service, every state has only one next state, but in non-deterministic one, it has a set of next states. One of sources for non-determinism is the *opaque data* in WS-BPEL (for the details, see Sections 8.4 and 13.1.3 in [21]). For instance, consider the following BPEL code:

```
<copy>
   <opaqueFrom/>
   <to variable="isAvailable"/>
</copy>
```

The `<copy>` statement copies a type-compatible value from the source ("from-spec") to the destination ("to-spec"). In many cases, a value or a variable appears in from-spec, and a variable is used in to-spec; the value (of the variable) in from-spec is assigned to the variable in to-spec. However, in the above case, a special tag `<opaqueFrom/>` in from-spec allows *any* value in the value space of the target variable (i.e, `isAvailable`) to be non-deterministically assigned to `isAvailable`; e.g., if the type of `isAvailable` is Boolean, this statement introduces two next states—in one state, `isAvailable` is *true*, and in the other `isAvailable` is *false*. *Any-Order* and *Choice* in OWL-S also induce non-determinism in the control flow of web services.

The second classification is *complete information*/*incomplete information*. The former is a special case of the WSC problem where a coordinator web service to be constructed knows the exact state of given web services at runtime (for this, all the variables web services contain should be output variables). The later is a general WSC problem where a coordinator web service knows only the values of output variables of web services. For instance, a web service for an airline reservation has an internal variable, `isAvailable`. A coordinator web service, however, should control this web service without the knowledge of the value of `isAvailable` during runtime. In the above problem settings, non-deterministic web services are more natural since non-determinism is allowed in WS-BPEL [21] and OWL-S [28]. Also, note that the complete information implies a rather unrealistic setting where all internal variables of $W$ should be exposed to outside as output variables. We then show that (1) solving the composition problem of *deterministic* web services based on *complete information* is **PSPACE-complete**; (2) solving the composition problem of *deterministic* web services based on *incomplete information* is **EXPSPACE-complete**; (3) solving the composition problem of *non-deterministic* web services on *complete information* is **EXP-complete** and (4) solving the composition problem of *non-deterministic* web services on *incomplete information* (which is the most general case) is **2-EXP-complete**. Table 1 presents the summary of our results. The transition from "complete information" to "incomplete information" induces exponential increment in state space. In the deterministic (non-deterministic, resp.) WSC problem, the complexity increases from PSPACE-complete (EXP-complete, resp.) to EXPSPACE-complete (2-EXP-complete, resp.). On the other hand, the transition from "deterministic" to "non-deterministic" induces complexity increment from a deterministic complexity to the corresponding alternating complexity. In the complete information (incomplete information, resp.) WSC problems, the complexity increases from PSPACE-complete (EXPSPACE-complete, resp.) to APSPACE-complete = EXP-complete (AEXPSPACE-complete = 2-EXP-complete, resp.). Our results suggest that it would be worthwhile studying efficient approximation solutions to the WSC problem.

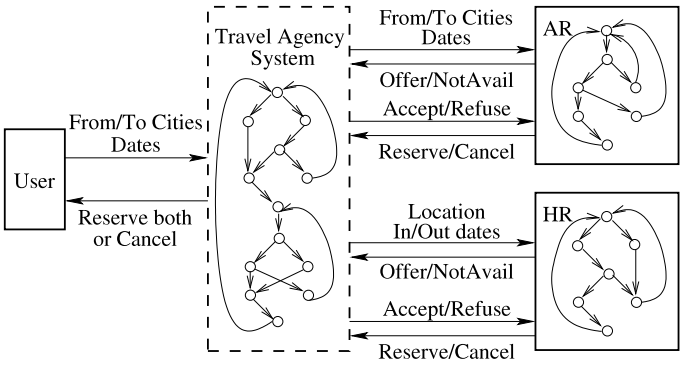



**Fig. 1.** Travel agency system.

### *1.1. Example: travel agency system*

As an example of web services, 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, suppose that there exist only an airline reservation (AR) web service and a hotel reservation (HR) web service separately. Clearly, we want to combine these two web services rather than implementing a new one. One way to combine them is to automatically construct a coordinator web service which communicates with each web service to book up a flight ticket and a hotel room both.

Fig. 1 illustrates this example. AR service receives a request including departing/returning dates, an origin and a destination, and then checks if the number of available seats for flights is greater than 0. If so, it returns the flight information and its price; otherwise, it returns "Not Available". Once offering the price, it waits for "Accept" or "Refuse" from its environment (in this case, a coordinator to be constructed). According to the answer, it processes the reservation. Likewise, HR service is requested with check-in/check-out dates and a location, and then checks the number of available rooms in an appropriate hotel. If there are available accommodations, it returns the room information and its price; otherwise, it returns "Not Available". AR then processes a reply "Accept" or "Refuse" from its environment.

The coordinator web service to be constructed receives from a user a request including departing/returning dates, an origin and a destination and, tries to achieve a goal, "reserve a flight ticket and a hotel room both OR cancel it", by controlling these two web services. For every output from AR and HR, the coordinator has to decide *one input* to them as the next action based on *only* output values (since in runtime it cannot access the internal variables in AR and HR, e.g., the number of available seats in flights), and it should accomplish the aim eventually. The coordinator can be represented by a deterministic state-transition system obviously.

### *1.2. Organization*

The rest of this paper is organized as follows. In Section 2, we discuss related work with this paper. Next, in Section 3 we lay out the notions and the problem we consider in this paper. Then we show the computational complexity for the composition problem of deterministic (non-deterministic) web services in Section 4 (Section 5, respectively). Finally, we give a few discussions and conclusions in Section 6.

## 2. Related work

In the research of Web services, substantial progress has already been made to provide languages for interoperation between heterogeneous systems. Languages such as Simple Object Access Protocol (SOAP) [29], Web Services Description Language (WSDL) [30], Universal Description, Discovery, and Integration (UDDI) [20] define standards for service discovery, description and invocation. On these standards, ontology-based languages such as DAML-S [8] and OWL-S [28] effort to bring the semantics into the service descriptions for automating the use of Web Services. On the other hands, some works (e.g., Web Service Business Process Execution Language (WS-BPEL) [21] are focused on representing service compositions by means of a workflow process consisting of a set of activities.

In web service compositions, much research [19,23,13,22,5,18] has been carried out, but only a few use realistic models with incomplete information (partial observability). [23,22,5] have defined web service compositions with incomplete information, and presented algorithms and tools using their AI planning techniques. However, there is no study for the complexity of this setting with detailed proofs. Recently, Fan et al. [10] investigate the complexity of web service composition based on query rewriting using views, but they include only deterministic web services in their problem setting, which is more natural. Moreover, their study does not provide the complete proof. For composition based on semantics, Mrissa et al. [17] have studied a context-based mediation approach to solve semantic heterogeneities between composed Web services. For signature-level WSC, Yu et al. [32] have developed a broker-based architecture to facilitate the selection of QoS-based services. Their goal of service selection is to maximize an application-specific utility function under the end-to-end

QoS constraints. Yu and Bouguettaya [31] have proposed a query algebra to generate Service Execution Plans that users can use to directly access services. A relevant line of web service composition is the formal verification for composite services. There are some research [11,14] for analyzing interactions of composite web services in BPEL processes. Beyer et al. [6] have proposed an interface language for specifying web service interfaces. Using this language, they present compatibility checking and substitutivity checking.

Finally, the WSC problem is related to AI planning under partial observation [12,4,25,16]. Herzig et al. [12] have proposed a dynamic logic EDL for planning under partial observability. In [4], a fully automatic planning tool MBP has been developed for this setting based on belief states. The complexity of planning under partial observability has been studied in [25]. Moffitt [16] has explored a means to both model and reason about partial observability within the scope of constraint-based temporal reasoning.

## 3. Preliminaries: behavioral description-based web service composition

**Definition 1** (*Web Service*)**.** A *web service* $w$ is a 5-tuple $(X, X^I, X^O, Init, T)$ with the following components:

- $X$ is a finite set of *variables* that $w$ controls. Every variable $x \in X$ has a finite domain (e.g., Boolean, bounded integers, or enumerated types). A state $s$ of $w$ is a valuation for every variable in $X$. We denote a set of all states[1] as $S$.
- $X^I$ is a finite set of *input variables* that $w$ reads from its environment; $X \cap X^I = \emptyset$, and every variable $x \in X^I$ has a finite domain. A state *in* for inputs is a valuation for every variable in $X^I$. We denote a set of all 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.
- $T(X, X^I, X')$ is a *transition predicate* over $X \cup X^I \cup X'$. 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 successor states. $T(s, in, s')$ is *true* iff $s'$ can be a next state when the input $in \in S^I$ is received at the state $s$. $T$ can define a *non-deterministic* transition relation. We assume that the transition relation is *total*; for every $s \in S$ and $in \in S^I$, there exists at least one $s'$ such that $T(s, in, s') = true$. □

**Definition 2** (*Deterministic/Non-deterministic Web Service*)**.** A web service $w(X, X^I, X^O, Init, T)$ is deterministic if $T$ is deterministic; namely, if for every state $s$ and every input *in*, there exists only one next state $s'$ (i.e., $T(s, in, s') = true$). Otherwise, $w$ is non-deterministic. □

Given a state $s$ over $X$ (i.e., a valuation for all variables in $X$) and a variable $x \in X$, $s(x)$ is the value of $x$ in $s$. For a state $s$ over $X$ and a set of variables $Y \subseteq X$, let $s[Y]$ denote the valuation over $Y$ obtained by restricting $s$ to $Y$. For every $x \in X^I$, we add a special value *null* into its domain. If a web service receives *null* as the value for any input variable, it stays in the same state; formally, if $T(s, in, s') = true$ such that $in(x) = null$ for some $x \in X^I$, then $s = s'$. Note that the process model for any web service described in Semantic Web languages (e.g., WS-BPEL [21] or OWL-S [28]) can be easily transformed into our representation of Definition 1 without any information loss as long as it has only finite domain variables and no recursion.[2]

**Example 1.** Let us consider a simple version of a web service $w$ for the airline reservation (AR) in Fig. 1, and assume that clients can request (reserve or refuse) a flight ticket by an action *req* (*accept* or *refuse*, respectively). The web service $w(X, X^I, X^O, Init, T)$ can be represented with the following elements:

- $X = \{\texttt{state}, \texttt{available}, \texttt{reply}, \texttt{confirm}\}$ where `state` has the domain $\{s_1, s_2\}$, `available` is Boolean, `reply` has the domain {*undecided*, *offer*, *notAvailable*}, and `confirm` has the domain {*undecided*, *reserve*, *cancel*}.
- $X^I = \{\texttt{action}\}$ where `action` has the domain {*req*, *accept*, *refuse*}.
- $X^O = \{\texttt{reply}, \texttt{confirm}\}$.
- $Init(X) \equiv (\texttt{state} = s_1) \wedge (\texttt{reply} = undecided) \wedge (\texttt{confirm} = undecided)$.
- $T(X, X^I, X') \equiv$

$$
\begin{aligned}
&(((\texttt{state} = s_1) \wedge (\texttt{action} = req) \wedge (\texttt{available} = true)) \rightarrow \\
&\qquad ((\texttt{state}' = s_2) \wedge (\texttt{reply}' = offer))) \quad \wedge \\
&(((\texttt{state} = s_1) \wedge (\texttt{action} = req) \wedge (\texttt{available} = false)) \rightarrow \\
&\qquad ((\texttt{state}' = s_1) \wedge (\texttt{reply}' = notAvailable))) \quad \wedge \\
&(((\texttt{state} = s_2) \wedge (\texttt{action} = accept)) \rightarrow \\
&\qquad ((\texttt{state}' = s_1) \wedge (\texttt{confirm}' = reserve))) \quad \wedge \\
&(((\texttt{state} = s_2) \wedge (\texttt{action} = refuse)) \rightarrow \\
&\qquad ((\texttt{state}' = s_1) \wedge (\texttt{confirm}' = cancel))). \quad \square
\end{aligned}
$$

**Definition 3** (*Set of Web Services*)**.** In the WSC problem, given a set of available web services, $W$, every web service in $W$ communicates only with their coordinator but not with each other. Based on this assumption, given a set $W = \{w_1, \ldots, w_n\}$ of web services where each $w_i = (X_i, X_i^I, X_i^O, Init_i, T_i)$, and $X_i$ and $X_i^I$ are disjoint with each other $X_j$ and $X_j^I$, respectively, $W$

1 The set $S$ of states is not encoded as the part of the web service.

2 For this translation, see [19].

also can be represented by a tuple $(X, X^I, X^O, Init, T)$[3] where:

- $X = X_1 \cup \cdots \cup X_n$.
- $X^I = X_1^I \cup \cdots \cup X_n^I$.
- $X^O = X_1^O \cup \cdots \cup X_n^O$.
- $Init(X) = Init_1(X_1) \wedge \cdots \wedge Init_n(X_n)$.
- $T(X, X^I, X') = T_1(X_1, X_1^I, X_1') \wedge \cdots \wedge T_n(X_n, X_n^I, X_n')$. □

Since a *coordinator web service* is also a web service, we note it as a 5-tuple web service, $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$. In what follows, we use $s$ as a state of $W$ and $sc$ as a state of a coordinator $c$. 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 coordinator state $sc$ and input $in$, there exists only one next coordinator state $sc'$ such that $T_c(sc, in, sc') = true$. Since non-deterministic coordinators suggests a set of (non-unique) inputs during runtime, it requires extra computation to select one of them, and is thus less desirable.

**Example 2.** Consider a simple coordinator web service $c$ that communicates with AR in Example 1. The coordinator web service $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ can be represented with the following elements:

- $X_c = \{\texttt{c\_state}, \texttt{action}\}$ where `c_state` has the domain $\{s_1, s_2\}$, and `action` has the domain $\{req, accept, refuse\}$.
- $X_c^I = \{\texttt{reply}, \texttt{confirm}\}$ where `reply` has the domain $\{undecided, offer, notAvailable\}$, and `confirm` has the domain $\{undecided, reserve, cancel\}$.
- $X_c^O = \{\texttt{action}\}$.
- $Init_c(X_c) \equiv (\texttt{c\_state} = s_1) \wedge (\texttt{action} = req)$.
- $T_c(X_c, X_c^I, X_c') \equiv$

$$
\begin{aligned}
&(((\texttt{c\_state} = s_1) \wedge (\texttt{reply} = offer)) \rightarrow \\
&\qquad ((\texttt{c\_state}' = s_2) \wedge (\texttt{action}' = accept))) \quad \wedge \\
&(((\texttt{c\_state} = s_1) \wedge (\texttt{reply} = notAvailable)) \rightarrow \\
&\qquad ((\texttt{c\_state}' = s_1) \wedge (\texttt{action}' = req))) \quad \wedge \\
&(((\texttt{c\_state} = s_2) \wedge (\texttt{confirm} = reserve)) \rightarrow \\
&\qquad ((\texttt{c\_state}' = s_1) \wedge (\texttt{action}' = req))) \quad \wedge \\
&(((\texttt{c\_state} = s_2) \wedge (\texttt{confirm} = cancel)) \rightarrow \\
&\qquad ((\texttt{c\_state}' = s_1) \wedge (\texttt{action}' = req))).
\end{aligned}
$$

□

**Definition 4** (*Execution Tree*)**.** Given a set of web services $W = (X, X^I, X^O, Init, T)$ and a coordinator $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ (in what follows, we assume that $X^I = X_c^O$ and $X^O = X_c^I$), we can define an *execution tree*, denoted by $W||c$, which represents the composition of $W$ and $c$ as follows:

- Each node $(s, sc)$ in $W||c$ is in $S \times S_c$.
- The root node is some $(s, sc)$ such that $Init(s) = true$ and $Init_c(sc) = true$.
- Each node $(s, sc)$ has a set of child nodes, $\{(s', sc') \mid T(s, in, s') = true, in = sc[X^I], T_c(sc, in_c, sc') = true, in_c = s'[X^O]\}$. □

Intuitively, the web services $W$ proceed, by receiving the input $in$ from the current state $sc$ of the coordinator, from $s$ to the next state $s'$, and then the coordinator proceeds, by receiving the input $in_c$ from the new state $s'$ of the web services, from $sc$ to the next state $sc'$. Even though the composition of $W$ and $c$ is defined above as synchronous communication, we can easily extend this model for *asynchronous* communication using $\tau$-transition (see [22]).

**Definition 5** (*Goal*)**.** A *goal* $G$ is a set of states we want to reach. In our problem setting, $G$ is represented by a Boolean formula over $X$. □

Given a set of web services $W$, a coordinator $c$, and a goal $G$, we define $W||c \models G$ if for every path $(s_0, sc_0)(s_1, sc_1) \cdots$ in the execution tree $W||c$, $s_i \in G$ for some $i \geq 0$; namely, every path from the initial node $(s_0, sc_0)$ reaches a goal state eventually.

**Definition 6** (*Web Service Composition Problem*)**.** The *web service composition (WSC)* problem that we focus on in this paper is, given a set of web services $W$ and a goal $G$, to construct a coordinator web service $c$ such that $W||c \models G$. □

In this paper, we assume that the problem size is the number of Boolean variables when we encode all variables in $X$ into Boolean variables. The reason is that the representation of a set of web services $W$ is typically much larger than the goal representation, and thus the size of $W$ is the computational bottleneck.

**Example 3.** Consider a WSC problem for Travel agency system in Fig. 1. In this example, we wish to reserve a flight ticket and a hotel room both. This requirement can be represented by $G \equiv (\texttt{flightConfirm} = reserve) \wedge (\texttt{hotelConfirm} = reserve)$. Now, given a set of web services $W = \{w_{AR}, w_{HR}\}$ and a goal $G$ above, a WSC problem is to automatically construct a coordinator web service $c$ such that $W||c \models G$. □

[3] A given set of web services can be replaced with a single web service. Since, in real WSC problems, multiple web services are given and they do not communicate with each other but only with a coordinator to be constructed, it is natural in the web service composition problems [23,22,5].

To study the computational complexity for web service composition problems, we define four variant WSC problems as follows:

(1) **Deterministic WSC with complete information**: Given $W(X, X^I, X^O, Init, T)$ such that $T$ is deterministic and $X = X^O$ (i.e., $W$ contains no internal variable), and a goal $G$, construct a coordinator web service $c$ such that $W||c \models G$.
(2) **Deterministic WSC with incomplete information**: Given $W(X, X^I, X^O, Init, T)$ such that $T$ is deterministic and no restriction for $X^O$ (i.e., a coordinator can read only the output variables in $X^O$), and a goal $G$, construct a coordinator web service $c$ such that $W||c \models G$.
(3) **Non-deterministic WSC with complete information**: Given $W(X, X^I, X^O, Init, T)$ such that $T$ is non-deterministic and $X = X^O$, and a goal $G$, construct a coordinator web service $c$ such that $W||c \models G$.
(4) **Non-deterministic WSC with incomplete information**: Given $W(X, X^I, X^O, Init, T)$ such that $T$ is non-deterministic and no restriction for $X^O$, and a goal $G$, construct a coordinator web service $c$ such that $W||c \models G$. This case is the most general WSC problem.

## 4. Computational complexities for deterministic web services

In this section, we study the computational complexities (i.e., lower bounds and upper bounds) of two deterministic WSC problems, (1) and (2), defined in Section 3. That is, we show that the deterministic WSC problem with complete information is PSPACE-complete and the deterministic WSC problem with incomplete information is EXPSPACE-complete.

### 4.1. Deterministic WSC with complete information

**Theorem 1** (*Lower Bound*). *The deterministic WSC problem with complete information is PSPACE-hard.* □

**Core idea of proof:** The proof is to simulate a deterministic Turing machine (DTM) with a polynomial space bound. That is, for any DTM $D$ and an input string $\sigma$, we can construct a deterministic WSC problem with complete information in polynomial time such that $D$ accepts $\sigma$ if and only if there exists a coordinator to satisfy a goal. The WSC problem instance has variables which represent the state, tape, and head position of DTM. If a coordinator provides an input corresponding to the current configuration of DTM, the WSC instance simulates one step of the DTM in each transition. The goal $G$ encodes all configurations where the simulation reaches an accepting state of DTM. First, we show how to construct such a WSC problem instance that simulates the computation of any PSPACE Turing machine. We then prove the implications $\rightarrow$ and $\leftarrow$ using Lemmas 1 and 2, respectively.

**Details of proof:** Given a DTM $D = (Q, \Sigma, q_0, \delta)$ with polynomial space bound $p(n)$ and an input string $\sigma = a_1 \cdots a_n$ (where $n = |\sigma|$), we can construct a WSC problem instance with $W(X, X^I, X^O, Init, T)$ and a goal $G$ (where $T$ is deterministic and $X = X^O$) which have a polynomial size in the size of the description of $D$ and $\sigma$ as follows. The set $X$ of variables includes the following variables:

- *state* represents the current state of $D$ with the domain, $\{q \mid q \in Q\}$.
- For $1 \leq i \leq p(n) + 1$, the variable $cell_i$ has the contents of the $i$-th tape cell; its domain is $\Sigma \cup \{\#\}$.
- *hd* describes the R/W head position; its domain is $\{1, \ldots, p(n) + 1\}$.

The set of input variables is $X^I = \{input\}$ where the domain of *input* is $\{(q, i, a) \mid q \in Q, 0 \leq i \leq p(n) + 1, a \in \Sigma\}$. The set $X^O$ of output variables equals to $X$ since this is the problem with complete information. As the initial configuration of $D$, the initial state predicate $Init(X)$ is $(state = q_0) \wedge \bigwedge_{1 \leq i \leq n}(cell_i = a_i) \wedge \bigwedge_{n < i \leq p(n)+1}(cell_i = \#) \wedge (hd = 1)$. Note that the input string is $\sigma = a_1 \cdots a_n$. To simulate the DTM $D$, the transition predicate $T$ strictly follows $\delta$ of $D$; that is, $T(X, X^I, X') \equiv ((hd = p(n) + 1) \rightarrow T_V) \wedge ((hd \neq p(n) + 1) \rightarrow T_N)$ with the following sub-formulas:

- $T_V \equiv (state' = state) \wedge \bigwedge_{1 \leq i \leq p(n)+1}(cell' = cell) \wedge (hd' = hd)$
- $T_N \equiv \bigwedge_{q \in Q, 1 \leq i \leq p(n), a \in \Sigma}(((state = q) \wedge (hd = i) \wedge (cell_i = a) \wedge (input = (q, i, a))) \rightarrow ((state' = q') \wedge (cell'_i = a') \wedge \bigwedge_{j \neq i}(cell'_j = cell_j) \wedge (hd' = hd + \Delta)))$

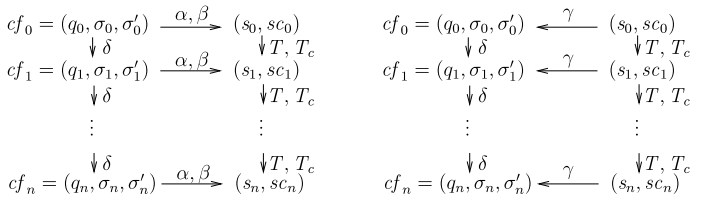
where $q'$ and $a'$ are obtained from $\delta(q, a) = (q', a', m')$, and $\Delta = -1$ if $m' = \mathcal{L}$, $\Delta = 0$ if $m' = \mathcal{N}$ and $\Delta = 1$ if $m' = \mathcal{R}$. Note that the value of the variable *input* is provided by a coordinator $c$. Finally, we have a goal, $G = \bigvee_{q \in F} state = q$.

If the DTM $D$ violates the space bound, *hd* has the value $p(n) + 1$, and after this point we cannot reach goal states since $W$ stays in the same state forever by $T_V$.

**Lemma 1.** *If $\sigma \in L(D)$, then there exists a coordinator $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ such that $W||c \models G$.* □

**Proof.** Let us assume that the configuration sequence of $D$ with respect to the input string $\sigma$ is $cf_0 \cdots cf_n$ where each $cf_i = (q_i, \sigma_i, \sigma'_i)$. $\sigma \in L(D)$ implies that the configuration sequence $(q_0, \sigma_0, \sigma'_0) \cdots (q_n, \sigma_n, \sigma'_n)$ reaches a final state; i.e., $q_n \in F$. We can construct such a coordinator $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ with the following elements:

- $X_c = \{input\}$; remark that the domain of *input* is $\{(q, i, a) \mid q \in Q, 0 \leq i \leq p(n) + 1, a \in \Sigma\}$.




(a) Mapping from configurations to nodes. (b) Mapping from nodes to configurations.

**Fig. 2.** Mapping between DTM and $W||c$ for Theorem 1.

- $X_c^I = X$.
- $X_c^O = \{input\}$.
- $Init_c(X_c) \equiv (input = (q_0, 1, \sigma[1]))$ where $q_0$ is the initial state of $D$ and $\sigma[1]$ is the first symbol of the input string $\sigma$.
- $T_c(X_c, X_c^I, X_c') \equiv \bigwedge_{q\in Q, 1\le i\le p(n)+1, a\in\Sigma}(((state = q) \wedge (hd = i) \wedge (cell_i = a)) \rightarrow (input' = (q, i, a)))$.

Now, we show that the configuration sequence $cf_0 \cdots cf_n$ is mapped to $(s_0, sc_0) \cdots (s_n, sc_n)$. Fig. 2(a) illustrates this mapping. We can easily find two mapping functions, $\alpha$ and $\beta$; $\alpha$ maps a configuration $cf$ to a state $s$ of web services $W$, and $\beta$ maps $cf$ to a state $sc$ of the coordinator $c$. Since the transition predicate $T$ of $W$ and transition predicate $T_c$ of $c$ strictly follow the transition $\delta$ of the DTM $D$, the execution path $(s_0, sc_0) \cdots (s_n, sc_n)$ based on $T$ and $T_c$ agrees with $(\alpha(cf_0), \beta(cf_0)) \cdots (\alpha(cf_n), \beta(cf_n))$. Hence, the execution path $(s_0, sc_0) \cdots (s_n, sc_n)$ also reaches a goal $G$ since $q_n \in F$. Finally, $W||c \models G$. □

**Lemma 2.** *If there exists a coordinator c such that $W||c \models G$, then $\sigma \in L(D)$.* □

**Proof.** The fact that there exists a coordinator $c$ such that $W||c \models G$ means that every path $(s_0, sc_0)(s_1, sc_1) \cdots (s_n, sc_n)$ reaches a goal state. We then construct a configuration sequence $cf_0 \cdots cf_n$ for the DTM $D$ that corresponds to the execution path. Fig. 2(b) illustrates this mapping. Again, we can easily find a mapping function $\gamma$ which maps a state $s$ of web services $W$ to a configuration $cf$ of $D$. Since $T$ and $T_c$ strictly follow the transition $\delta$ of the DTM $D$, the configuration sequence $cf_0 \cdots cf_n$ for the DTM $D$ is consistent with $\gamma(s_0) \cdots \gamma(s_n)$. Hence, the configuration sequence reaches an accepting configuration $cf_n = (q_n, \sigma_n, \sigma'_n)$ such that $q_n \in F$ since $s_n \in G$. Therefore, $\sigma \in L(D)$. □

**Theorem 2** (*Upper Bound*)**.** *The deterministic WSC problem with complete information is in PSPACE.* □

**Proof.** We first show that the problem is in NPSPACE. Since the transition of $W$ is deterministic, we just need to find a path from the initial state of $W$ to a goal state. Then, we can easily construct a coordinator from the path in polynomial space. The solution path can be non-deterministically chosen, and each state of the path is encoded by $n$ variables where $n = |X|$. If there exists a solution path, its length is at most $2^n$ since the number of states is at most $2^n$. Therefore, at most $2^n$ non-deterministic choices are required. This process can be encoded in polynomial space. Hence, the problem is NPSPACE. Finally, since NPSPACE = PSPACE [26], the deterministic WSC problem with complete information is in PSPACE. □

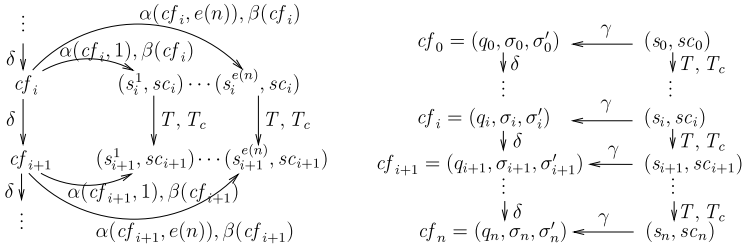
### 4.2. Deterministic WSC with incomplete information

**Theorem 3** (*Lower Bound*)**.** *The deterministic WSC problem with incomplete information is EXPSPACE-hard.* □

**Core idea of proof:** The proof is to simulate a DTM with exponential space bound. An important difference from the complete information problem case is that we are *not* allowed to have a variable for each tape cell since the number of tape cells is exponential and the reduction could not be polynomial. Instead of including an exponential number of variables $cell_i$, we have one variable *cell* and its index *idx*. The trick is to establish that if the index matches the current head position, $W$ should simulate the DTM $D$ (otherwise, $W$ keeps the same symbol in *cell*), and to force the above to be satisfied universally for every index *idx*. The intuition underlying our trick is that we map the requirement of the exponential length tape into incomplete information (i.e., the coordinator should control $W$ with knowledge for only *cell* rather than all the tape cells).

**Details of proof:** Given a DTM $D = (Q, \Sigma, q_0, \delta)$ with exponential space bound $e(n)$ and an input string $\sigma = a_1 \cdots a_n$ (where $n = |\sigma|$), we can construct a deterministic WSC problem instance with $W(X, X^I, X^O, Init, T)$ and a goal $G$ (where $T$ is deterministic and $X = X^O$) as follows. The set $X$ of variables includes the following variables:

- *state*; its domain is $\{q \mid q \in Q\}$.
- *idx*; its domain is $\{1, \ldots, e(n)\}$.
- *cell* 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 $\lceil \log_2(e(n) + 1) \rceil$ bits.




(a) Mapping from configurations to nodes. (b) Mapping from nodes to configurations.

**Fig. 3.** Mapping between DTM and $W||c$ for Theorem 3.

- *lsb* represents the symbol written by the head in the last step; it has a domain, $\Sigma \cup \{\#\}$.
- *lmv* represents the last head movement; it has a domain, $\{\mathcal{L}, \mathcal{N}, \mathcal{R}\}$.

The set $X^I$ is $\{input\}$ where the domain of *input* is $\{(q, a) \mid q \in Q, a \in \Sigma\}$. The set $X^O$ is $\{state, cell\}$. $Init(X)$ is $(state = q_0) \wedge ((idx \leq |\sigma|) \Leftrightarrow (cell = \sigma[idx])) \wedge ((idx > |\sigma|) \Leftrightarrow (cell = \#)) \wedge (hd = 1)$ where $\sigma$ is the input string. The initial predicate allows any value for *idx*, and the value for *cell* is determined by *idx*. The transition predicate $T(X, X^I, X')$ is $((hd = e(n) + 1) \rightarrow T_V) \wedge ((hd \neq e(n) + 1) \rightarrow T_N)$ where:

- $T_V \equiv (state' = state) \wedge (idx' = idx) \wedge (cell' = cell) \wedge (hd' = hd) \wedge (lsb' = lsb) \wedge (lmv' = lmv)$
- $T_N \equiv \bigwedge_{q \in Q, a \in \Sigma}(((state = q) \wedge ((hd = idx) \rightarrow (cell = a)) \wedge (input = (q, a))) \rightarrow$
  $(((hd = idx) \rightarrow (cell' = a')) \wedge ((hd \neq idx) \rightarrow (cell' = cell)) \wedge (state' = q') \wedge (idx' = idx) \wedge (hd' = hd + \Delta) \wedge (lsb' = a') \wedge (lmv' = m')))$

where $q'$, $a'$ and $m'$ are obtained from $\delta(q, a) = (q', a', m')$, and $\Delta = -1$ if $m' = \mathcal{L}$, $\Delta = 0$ if $m' = \mathcal{N}$ and $\Delta = 1$ if $m' = \mathcal{R}$. Finally, we have a goal, $G = \bigvee_{q \in F} state = q$.

If the DTM $D$ violates the space bound, *hd* has the value $e(n) + 1$, and after this point we cannot reach goal states since $W$ stays in the same state forever by $T_V$.

**Lemma 3.** *If $\sigma \in L(D)$, then there exists a coordinator $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ such that $W||c \models G$.* □

**Proof.** We can construct such a coordinator $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ with the following elements:

- $X_c = \{input\}$; remark that the domain of *input* is $\{(q, a) \mid q \in Q, a \in \Sigma\}$.
- $X_c^I = \{state, cell\}$.
- $X_c^O = \{input\}$.
- $Init_c(X_c) \equiv (input = (q_0, \sigma[1]))$ where $q_0$ is the initial state of $D$ and $\sigma[1]$ is the first symbol of the input string $\sigma$.
- $T_c(X_c, X_c^I, X_c') \equiv \bigwedge_{q \in Q, a \in \Sigma}(((state = q) \wedge (cell = a)) \rightarrow (input' = (q, a)))$.

Then, we show that the configuration sequence $(q_0, \sigma_0, \sigma_0') \cdots (q_n, \sigma_n, \sigma_n')$ is mapped to an execution tree $W||c$. Fig. 3(a) shows our mapping. For this mapping, we have two mapping functions, $\alpha$ and $\beta$; $\alpha$ maps a configuration $cf$ in the configuration sequence to a state $s$ of web services $W$, and $\beta$ maps $cf$ to a state $sc$ of the coordinator $c$. First, given a configuration $cf = (q, \sigma_1, \sigma_2)$ and a tape index $1 \leq i \leq e(n)$, we have a corresponding state $s = \alpha(cf, i)$ of $W$ such that

- $s(state) = q$.
- $s(idx) = i$.
- $s(cell) = (\sigma_1\sigma_2)[i]$ if $i \leq \sigma_1\sigma_2$; otherwise, $s(cell) = \#$.
- $s(hd) = |\sigma_1|$.

Next, for each configuration $cf = (q, \sigma_1, \sigma_2)$, we have a corresponding state $sc = \beta(cf)$ of $c$ such that $sc(input) = (q, \sigma_1[|\sigma_1|])$. Now, by induction, we can show that if $cf$ in the configuration sequence reaches an accepting configuration by $d$ steps, then every path from any $(s, sc)$ in $W||c$, which corresponds to $cf$ (i.e., $(s, sc) = (\alpha(cf, i), \beta(cf))$), reaches a goal state in $d$ steps. □

**Lemma 4.** *If there exists a coordinator $c$ such that $W||c \models G$, then $\sigma \in L(D)$.* □

**Proof.** We show that a configuration sequence $(q_0, \sigma_0, \sigma_0') \cdots (q_n, \sigma_n, \sigma_n')$ of $D$ corresponding to the execution path can be constructed to reach an accepting configuration. First, we have the initial configuration $(q_0, \sigma_0, \sigma_0')$ such that $q_0$ is the initial state of $D$, $\sigma_0 = \sigma[1]$ (where $\sigma$ is the input string), and $\sigma_0' = \sigma[2] \cdots \sigma[n]$. Then, we can construct $i$-th configuration

$cf_i = (q_i, \sigma_i, \sigma_i')$ from the previous configuration $cf_{i-1} = (q_{i-1}, \sigma_{i-1}, \sigma_{i-1}')$ and the state $s_i$ of $W$. Fig. 3(b) illustrates this mapping. For the mapping, we have a mapping function $\gamma$ which maps $s_i$ and $cf_{i-1}$ to a configuration $cf_i$ of $D$. That is, for each state $s_i$, we have a corresponding configuration $cf_i = \gamma(s_i, cf_{i-1}) = (q_i, \sigma_i, \sigma_i')$ where $q_i = s_i(state)$, $\sigma_i$ and $\sigma_i'$ can be generated with $\sigma_{i-1}$, $\sigma_{i-1}'$, $s_i(lsb)$ and $s_i(lmv)$.

Now, by induction we can show that if a node $(s, sc)$ in the execution path of $W||c$ reaches a goal in $d$ steps, then the corresponding configuration $cf$ also reaches an accepting configuration in $d$ steps. □

**Theorem 4** (*Upper Bound*). *The deterministic WSC problem with incomplete information is in EXPSPACE.* □

**Proof.** We first show that this problem is in NEXPSPACE. As Theorem 2, since the transition of $W$ is deterministic, we just need to find a path from the initial state of $W$ to a goal state. The main difference is that a coordinator web service is not able to identify the exact state of target web services due to incomplete information. We model this uncertainty by using a *belief state* [4], which is a set of *possible* states of target web services but *indistinguishable*. The coordinator should make a decision based on the current belief state, and the number of possible belief states is $2^{2^n}$ where $n = |X|$. Now, the solution path can be non-deterministically chosen, and each belief state of the path is encoded by at most $n$ variables. If there exists a solution path, its length is at most $2^{2^n}$ since the number of belief states is at most $2^{2^n}$. Therefore, at most $2^{2^n}$ non-deterministic choices are required. This process can be encoded in $2^n$ space. Hence, the problem is NEXPSPACE. Finally, since NEXPSPACE = EXPSPACE [26], the deterministic WSC problem with incomplete information is in EXPSPACE. □

## 5. Computational complexities for non-deterministic web services

In this section, we study the computational complexities of two non-deterministic WSC problems, (3) and (4), defined in Section 3. First, we show that the non-deterministic WSC problem with complete information (incomplete information, respectively) is ASPACE-hard (resp. AEXPSPACE-hard), and by Chandra et al.'s result[4] the problem is EXP-hard (2-EXP-hard, respectively). Our proofs for lower bounds use reductions from alternating Turing machines (ATMs) [1,2]. The main difference between ATMs and DTMs is that the transition function of ATMs is non-deterministic (i.e., for each state and input symbol, ATMs have a set of successor states) and there is alternation for states. States of ATMs are labeled by ∃ or ∀; to accept an input string, ∃-states are required to have at least one successor state to reach an accepting state while ∀-states need that all the successors reach an accepting state. Then, we present an upper bound for each problem.

### 5.1. Non-deterministic WSC with complete information

**Theorem 5** (*Lower Bound*). *The non-deterministic WSC problem with complete information is EXP-hard.* □

**Core idea of proof:** The proof is to simulate an ATM with a polynomial space bound. That is, for any ATM $A$ and an input string $\sigma$, we can construct a non-deterministic WSC problem with complete information in polynomial time such that $A$ accepts $\sigma$ if and only if there exists a coordinator to satisfy a goal. The WSC problem instance has variables which represent the state, tape, head position and alternation label of ATM. If a coordinator provides an input corresponding to the current configuration of ATM, the WSC instance simulates one step of the ATM in each transition. The goal $G$ encodes all configurations in which the simulation reaches an accepting state of the ATM. First, we show how to construct such a WSC problem instance that simulates the computation of any ATM. We then prove the implications → and ← using Lemmas 5 and 6, respectively.

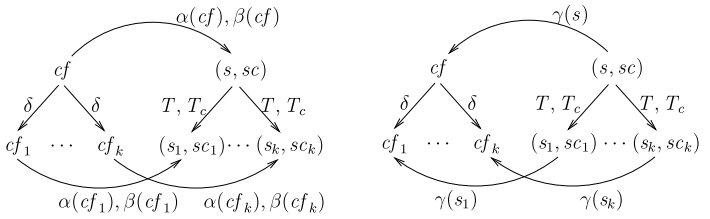
**Details of proof:** Given an ATM $A = (Q, \Sigma, q_0, \delta, l)$ with polynomial space bound $p(n)$ and an input string $\sigma = a_1 \cdots a_n$ (where $n = |\sigma|$), we can construct a WSC problem instance with $W(X, X^I, X^O, Init, T)$ of web services and a goal $G$ which have a polynomial size in the size of the description of $A$ and $\sigma$ as follows. The set $X$ of variables includes the following variables:

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

The set of input variables is $X^I = \{input\}$ where the domain of *input* is $\{\mathcal{A}_{(q,i,a)} \mid q \in Q, l(q) = \forall, 0 \le i \le p(n) + 1, a \in \Sigma\} \cup \{\mathcal{E}_{(q,i,a,j)} \mid q \in Q, l(q) = \exists, 0 \le i \le p(n) + 1, a \in \Sigma, 0 \le j \le |\delta(a, q)|\}$. The set $X^O$ of output variables equals to $X$ since this problem is the complete information problem. As the initial configuration of $A$, the initial state predicate $Init(X)$ is $(state = q_0) \wedge \bigwedge_{1 \le i \le n}(cell_i = a_i) \wedge \bigwedge_{n < i \le p(n)+1}(cell_i = \#) \wedge (hd = 1) \wedge (label = l(q_0))$. Note that the input string is $\sigma = a_1 \cdots a_n$. The transition predicate $T(X, X^I, X')$ is $((hd = p(n) + 1) \rightarrow T_\forall) \wedge (((hd \neq p(n) + 1) \wedge (label = \forall)) \rightarrow T_\forall) \wedge (((hd \neq p(n) + 1) \wedge (label = \exists)) \rightarrow T_\exists)$ with the following sub-formulas:

- $T_V \equiv (state' = state) \wedge \bigwedge_{1 \le i \le p(n)+1}(cell_i' = cell_i) \wedge (hd' = hd) \wedge (label' = label)$

[4] APSPACE = EXP, and AEXPSPACE = 2-EXP [7].




(a) Mapping from configurations to nodes. (b) Mapping from nodes to configurations.

**Fig. 4.** Mapping between ATM and $W||c$ for Theorem 5.

- $T_\forall \equiv \bigwedge_{q\in Q, 1\le i\le p(n), a\in\Sigma}(((state = q) \wedge (hd = i) \wedge (cell_i = a) \wedge (input = \mathcal{A}_{(q,i,a)})) \rightarrow \bigvee_{1\le j\le k}((state' = q_j) \wedge (cell'_i = a_j) \wedge \bigwedge_{h\neq i}(cell'_h = cell_h) \wedge (hd' = hd + \Delta_j) \wedge (label' = l(q_j))))$
- $T_\exists \equiv \bigwedge_{q\in Q, 1\le i\le p(n), a\in\Sigma, 1\le j\le k}(((state = q) \wedge (hd = i) \wedge (cell_i = a) \wedge (input = \mathcal{E}_{(q,i,a,j)})) \rightarrow ((state' = q_j) \wedge (cell'_i = a_j) \wedge \bigwedge_{h\neq i}(cell'_h = cell_h) \wedge (hd' = hd + \Delta_j) \wedge (label' = l(q_j))))$

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 $\Delta_j = -1$ if $m_j = \mathcal{L}$, $\Delta_j = 0$ if $m_j = \mathcal{N}$ and $\Delta_j = 1$ if $m_j = \mathcal{R}$. Note that the value for the variable *input* is provided by a coordinator $c$. Finally, we have a goal, $G = \{s \in S \mid s(label) = accept\}$.

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 in the same state forever by $T_V$.

**Lemma 5.** *If $\sigma \in L(A)$, then there exists a coordinator $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ such that $W||c \models G$.* □

**Proof.** $\sigma \in L(A)$ means that the initial configuration of $A$ with respect to $\sigma$ is $d$-accepting ($d \geq 0$). Now, we show that there exists a coordinator $c$ such that for every path $(s_0, sc_0) \cdots (s_n, sc_n)$ in the execution tree $W||c$, $s_n \in G$. The coordinator to be constructed is $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ where $X_c = \{input\}$, $X_c^I = X$, and $X_c^O = \{input\}$.

When $A$ accepts $\sigma$, we can define an *accepting computation tree* $ACT_{(A,\sigma)}$ of $A$ with respect to $\sigma$ from its computation tree $\Upsilon$ as follows:

- For each configuration $cf = (q, \sigma_1, \sigma_2)$ in $\Upsilon$ such that $l(q) = \forall$, all the successor configuration are also included in $ACT_{(A,\sigma)}$. Note that if $cf$ is $d$-accepting, each successor is at most $(d-1)$-accepting.
- For each $cf = (q, \sigma_1, \sigma_2) \in \Upsilon$ such that $l(q) = \exists$ and $cf$ is $d$-accepting, only one successor configuration $cf'$ which is $(d-1)$-accepting is included in $ACT_{(A,\sigma)}$.

When $A$ and $\sigma$ are clear from the context, we drop the subscript $_{(A,\sigma)}$ and write $ACT$. Then, $ACT_{(A,\sigma)}$ is mapped to an execution tree $W||c$. Fig. 4(a) shows our mapping. For this mapping, we have two mapping functions, $\alpha$ and $\beta$; $\alpha$ maps a configuration $cf$ in $ACT$ to a state $s$ of web services $W$, and $\beta$ maps $cf$ to a state $sc$ 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 $1 \leq i \leq |\sigma_1\sigma_2|$, $s(cell_i) = \sigma_1\sigma_2[i]$, and for $|\sigma_1\sigma_2| < i \leq p(n) + 1$, $s(cell_i) = \#$.
- $s(hd) = |\sigma_1|$.
- $s(label) = l(q)$.

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

- If $l(q) = \forall$, then $sc(input) = \mathcal{A}_{(q,i,a)}$ 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, $sc(input) = \mathcal{E}_{(q,i,a,j)}$ where $i = |\sigma_1|$, $a = \sigma_1[i]$, and $j$ is the index of $cf'$.

According to $\alpha$ and $\beta$, we have an execution tree of $W||c$ where each node is $(\alpha(cf), \beta(cf))$. Then, by induction, we can show that if $cf$ in $ACT$ is $d$-accepting, every path from $(s, sc)$ corresponding to $(\alpha(cf), \beta(cf))$ in $W||c$ reaches a goal state. □

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

**Proof.** As shown in Section 3, the fact that there exists a coordinator $c$ such that $W||c \models G$ means that every path $(s_0, sc_0) \cdots (s_n, sc_n)$ from the initial node in the execution tree $W||c$ reaches a goal state eventually. Now, we show that an accepting computation tree $ACT$ for $A$ corresponding to $W||c$ can be constructed and the initial configuration is $d$-accepting.

For the mapping, we have a mapping function $\gamma$ which maps a state $s$ of web services $W$ to a configuration $cf$ of $A$. Fig. 4(b) shows this mapping by $\gamma$. For each state $s$ such that $s(state) = q$, $s(cell_i) = b_i$ where $1 \leq i \leq p(n) + 1$,

**Algorithm 1:** Algorithm for non-deterministic WSC with complete information

**Input** : A set $W$ of web services and a goal $G$.
**Output**: A coordinate web service $c$.

```
1  tree := InitializeSearchingTree(Init);
2  tree.root.result := undecided;
3  if (States(Init) ⊆ States(G)) then
4  |  tree.root.result := true;
5  end if
6  while (tree.root.result = undecided) do
7  |  node := SelectNode(tree);
8  |  childNodes := ExtendTree(tree, node);
9  |  if (CheckSuccess(childNodes)) then
10 |  |  node.result := true;
11 |  end if
12 |  else if (CheckFailure(childNodes)) then
13 |  |  node.result := false;
14 |  end if
15 |  PropagateResult(tree, node);
16 end while
17 if (tree.root.result = true) then
18 |  return ConstructCoordinator(tree);
19 end if
20 else return null;
```

$s(hd) = i$ and $s(label) = l(q)$, we have a corresponding configuration $cf = \gamma(s) = (q, \sigma_1, \sigma_2)$ such that $\sigma_1 = b_1 \cdots b_i$ and $\sigma_2 = b_{i+1} \cdots b_{j-1}$ where $j$ is the index of the first appearance of #.

Now, by induction, we can show that if among every path from a node $(s, sc)$ to a goal in $W||c$, the length of the longest one is $d$, the corresponding configuration $\gamma(s)$ is $d$-accepting. □

**Theorem 6** (*Upper Bound*)**.** *The non-deterministic WSC problem with complete information is in EXP.* □

**Proof.** We present a main procedure for this problem in Algorithm 1. The underlying idea of Algorithm 1 is to construct an *and–or searching tree* from initial states to goal states. That is, from any state of the tree, for non-determinism of output values of web services, we extend the tree with a set of child states via *and-edges*. In this case, all the child states should reach a goal state. For the coordinator selecting input values, we construct a set of child states via *or-edges*. In this case, at least one child is required to reach a goal state.

To initialize the and–or searching tree, Algorithm 1 first constructs a root state corresponding to the given initial predicate, *Init*, and assigns "*undecided*" to the result value for the root (lines 1–2). If the states corresponding to *Init* are already included in goal states, we assign "*true*" to the result value for the root (lines 3–5). Next (lines 6–16), until determining the result value for the root, we repeat: (1) select a node which is not determined yet as "*true*" or "*false*" (line 7); (2) extend the tree from the selected state by computing a set of possible successor states (line 8) and (3) check if the state can reach a goal state based on the and–or constraint (lines 9–14). Once we identify the result of each state, we propagate the result to its ancestor state (line 15). Finally, if the algorithm identifies the result of the root state as *true*, it constructs a coordinator web service from the tree, and returns the coordinator (lines 17–19). Otherwise, it returns *null* (line 20). Since the while loop with careful extension is executed only once per a state and each loop can be done in polynomial time, the complexity of the algorithm is $O(2^n)$ where $n$ is the number of variables in $W$. Therefore, the non-deterministic WSC problem with complete information is in EXP. □

### 5.2. Non-deterministic WSC with incomplete information

**Theorem 7** (*Lower Bound*)**.** *The non-deterministic WSC problem with incomplete information is 2-EXP-hard.* □

**Core idea of proof:** The proof is to simulate an ATM with exponential space bound. We first show that for any ATM $A$ and an input string $\sigma$, we can construct a non-deterministic WSC problem $W$ with incomplete information and a goal $G$. We then prove that if the ATM $A$ accepts $\sigma$, then we have a coordinator $c$ such that $W||c \models G$ by Lemma 7, and prove the other direction by Lemma 8. The main difference from the proof of Theorem 5 is that we are not allowed to have a variable for each tape cell since the reduction could not be done in polynomial. We, thus, employ a similar proof technique to Theorem 3; we have one variable *cell* and its index *idx*, instead of including an exponential number of variables $cell_i$. We then construct $W$ to satisfy that if the index matches the current head position, $W$ should simulate the ATM $A$ (otherwise, $W$ keeps the same symbol in *cell*).

**Details of proof:** Given an ATM $A = (Q, \Sigma, q_0, \delta, l)$ with exponential space bound $e(n)$ and an input string $\sigma = a_1 \cdots a_n$ (where $n = |\sigma|$), we can construct a WSC problem instance with $W(X, X^I, X^O, Init, T)$ and a goal $G$ as follows. The set $X$ of variables includes the following variables:

- *state*; its domain is $\{q \mid q \in Q\}$.
- *idx*; its domain is $\{1, \ldots, e(n)\}$.
- *cell* 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 $\lceil \log_2(e(n) + 1) \rceil$ 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, $\Sigma \cup \{\#\}$.
- *lmv* represents the latest head movement; it has a domain, $\{\mathcal{L}, \mathcal{N}, \mathcal{R}\}$.

The set $X^I$ is $\{input\}$ where the domain of *input* is $\{\mathcal{A}_{(q,a)} \mid q \in Q, l(q) = \forall, a \in \Sigma\} \cup \{\mathcal{E}_{(q,a,j)} \mid q \in Q, l(q) = \exists, a \in \Sigma, 0 \le j \le |\delta(a, q)|\}$. The set $X^O$ is $\{state, cell\}$. $Init(X)$ is $(state = q_0) \wedge ((idx \le |\sigma|) \Leftrightarrow (cell = a_{idx})) \wedge ((idx > |\sigma|) \Leftrightarrow (cell = \#)) \wedge (hd = 1) \wedge (label = l(q_0))$. The initial predicate allows any value for *idx*, and the value for *cell* is determined by *idx*. The transition predicate $T(X, X^I, X')$ is $((hd = e(n) + 1) \rightarrow T_V) \wedge (((hd = e(n) + 1) \wedge (label = \forall)) \rightarrow T_\forall) \wedge (((hd = e(n) + 1) \wedge (label = \exists)) \rightarrow T_\exists)$ with the following sub-formulas:

- $T_V \equiv (state' = state) \wedge (idx' = idx) \wedge (cell' = cell) \wedge (hd' = hd) \wedge (label' = label) \wedge (lsb' = lsb) \wedge (lmv' = lmv)$
- $T_\forall \equiv \bigwedge_{q \in Q, a \in \Sigma}(((state = q) \wedge ((hd = idx) \rightarrow (cell = a)) \wedge (input = \mathcal{A}_{(q,a)})) \rightarrow$
  $\bigvee_{1 \le j \le k}(((hd = idx) \rightarrow (cell' = a_j)) \wedge ((hd \ne idx) \rightarrow (cell' = cell)) \wedge (state' = q_j) \wedge (idx' = idx) \wedge (hd' = hd + \Delta_j) \wedge (label' = l(q_j)) \wedge (lsb' = a_j) \wedge (lmv' = m_j)))$
- $T_\exists \equiv \bigwedge_{q \in Q, a \in \Sigma, 1 \le j \le k}(((state = q) \wedge ((hd = idx) \rightarrow (cell = a)) \wedge (input = \mathcal{E}_{(a,q,j)})) \rightarrow$
  $(((hd = idx) \rightarrow (cell' = a')) \wedge ((hd \ne idx) \rightarrow (cell' = cell)) \wedge (state' = q_j) \wedge (idx' = idx) \wedge (hd' = hd + \Delta_j) \wedge (label' = l(q_j)) \wedge (lsb' = a_j) \wedge (lmv' = m_j)))$

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 $\Delta_j = -1$ if $m_j = \mathcal{L}$, $\Delta_j = 0$ if $m_j = \mathcal{N}$ and $\Delta_j = 1$ if $m_j = \mathcal{R}$. Finally, we have a goal, $G = \{s \in S \mid s(label) = accept\}$.

If the ATM $A$ violates the space bound, *hd* has the value $e(n) + 1$, and after this point we cannot reach goal states since $W$ stays in the same state forever by $T_V$.

**Lemma 7.** *If $\sigma \in L(A)$, then there exists a coordinator $c$ such that $W||c \models G$.* □

**Proof.** Given an ATM $A$ with an input string $\sigma$ 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 $X_c = \{input\}$, $X_c^I = \{state, cell\}$, and $X_c^O = \{input\}$. As in the proof of Lemma 5, 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 $\bigwedge_{q \in Q, a \in \Sigma}(((state = q) \wedge (cell = a)) \rightarrow (input' = \mathcal{A}_{(q,a)}))$. Otherwise, $\bigwedge_{q \in Q, a \in \Sigma}(((state = q) \wedge (cell = a)) \rightarrow (input' = \mathcal{E}_{(q,a,j)}))$ where $j$ is the index of the transition by which the ATM proceeds from the corresponding $\exists$-configuration to the next in the accepting computation tree $ACT_{(A,\sigma)}$. Similarly with $T_c$, we can define the initial predicate $Init_c$ as $((l(q_0) = \forall) \rightarrow (input = \mathcal{A}_{(q_0,a_1)})) \wedge ((l(q_0) = \exists) \rightarrow (input = \mathcal{E}_{(q_0,a_1,j)}))$ where $a_1$ is the first symbol of the input string $\sigma$ and $j$ is obtained as the above.

Now, we show that $ACT_{(A,\sigma)}$ is mapped to an execution tree $W||c$. For this mapping, we have two mapping functions, $\alpha$ and $\beta$; $\alpha$ maps a configuration $cf$ in $ACT$ to a state $s$ of web services $W$, and $\beta$ maps $cf$ to a state $sc$ of the coordinator $c$. Fig. 5 shows the proposed mapping by $\alpha$ and $\beta$. First, given a configuration $cf = (q, \sigma_1, \sigma_2)$ and a tape index $1 \le i \le e(n)$, we have a corresponding state $s = \alpha(cf, i)$ of $W$ such that

- $s(state) = q$.
- $s(idx) = i$.
- $s(cell) = \sigma_1\sigma_2[i]$ if $i \le \sigma_1\sigma_2$; otherwise, $s(cell) = \#$.
- $s(hd) = |\sigma_1|$.
- $s(label) = l(q)$.

The mapping function $\beta$ is the same as $\beta$ in Lemma 5. Now, we claim that if $cf$ in $ACT$ is $d$-accepting, then for every $1 \le i \le e(n)$ 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.

Finally, since the initial configuration of $ACT$ is $d$-accepting, every path from the initial node of $W||c$ reaches a goal state; that is, $W||c \models G$. □

**Lemma 8.** *If there exists a coordinator $c$ such that $W||c \models G$, then $\sigma \in L(A)$.* □

**Proof.** For the execution tree $W||c$, we construct an accepting computation tree $ACT_{(A,\sigma)}$. However, unlike Lemma 6, 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 *cell* and *lsb*. Now, our trick is to construct the computation tree by the top-down manner like the proof of Lemma 4. Even though

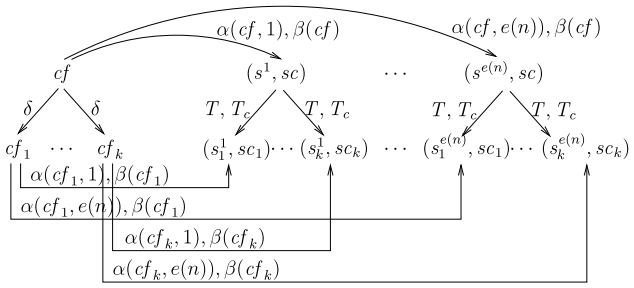



**Fig. 5.** Mapping from configurations of ATM to nodes of $W||c$ for Theorem 7.

the initial state of $W$ has only *cell* and *lsb*, we can construct the initial configuration as $cf = (q_0, a_1, \sigma')$ where the input string $\sigma = a_1\sigma'$. Given a predecessor configuration $cf_1 = (q_1, \sigma_1, \sigma_1')$ and a state $s$ of $W$ such that $s(state) = q$, $s(cell) = a$, $s(idx) = i$, $s(hd) = h$, $s(label) = l(q)$, and $s(lsb) = a'$, our mapping function $\gamma$ maps $s$ to a configuration $cf_2 = (q, \sigma_2, \sigma_2')$ where $|\sigma_2| = h$ and for $\sigma_2$ and $\sigma_2'$, $\sigma_2\sigma_2'$ is copied from $\sigma_1\sigma_1'$ except $(\sigma_2\sigma_2')[|\sigma_1|] = a'$. This mapping is similar to the one in Fig. 4(b).

Now, we claim that among every path from a node $(s, sc)$ to a goal in $W||c$, if the length of the longest one is $d$, the corresponding configuration $\gamma(s)$ is $d$-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.

Finally, since the initial node $(s, sc)$ of $W||c$ has $d$ (for some $d \geq 0$) as the length of the longest path to a goal, the initial configuration of $A$ is $d$-accepting. □

**Theorem 8** (*Upper Bound*)**.** *The non-deterministic WSC problem with incomplete information is in 2-EXP.* □

**Proof.** The basic idea of the procedure for this problem is the same as Algorithm 1 which is for the non-deterministic WSC problem with complete information. The main difference is that a coordinator web service is not able to identify the exact state of target web services due to incomplete information. As Theorem 4, we can model this uncertainty by using a *belief state* [4]. The coordinator should make a decision based on the current belief state, and the number of possible belief states is $2^{2^n}$ where $n = |X|$. Now, we can construct an *and–or searching tree* based on *belief states*. In this problem, since the while loop of Algorithm 1 is executed once per a belief state, the complexity of the procedure for this problem is $O(2^{2^n})$. Accordingly, the non-deterministic WSC problem with incomplete information is in 2-EXP. □

## 6. Conclusion and discussion

In this paper, we have formally defined the realistic model for behavioral description-based web service composition (WSC) problems, and studied the computational complexity of four variations of WSC problems. The main findings of this paper are as follows: (1) solving the composition problem of *deterministic* web services based on *complete information* is **PSPACE-complete**; (2) solving the composition problem of *deterministic* web services based on *incomplete information*) is **EXPSPACE-complete**; (3) solving the composition problem of *non-deterministic* web services on *complete information* is **EXP-complete** and (4) solving the composition problem of *non-deterministic* web services on *incomplete information* (which is the most general case) is **2-EXP-complete**.

Our findings suggest that much more efforts to find *alternative* solutions to the WSC problem be needed. For instance, using an approximation algorithm with near-polynomial time complexity to solve the WSC problem can be more beneficial in some applications. As a large number of web services become available and a fast real-time response to the composition is needed, one may not afford the exponential complexity of a solution. Another possibility to mitigate the complexity is via some reduction. For instance, by abstracting internal variables of web services, one can reduce the incomplete information WSC problem into the complete information WSC problem. Although the over-approximation induces that the solution may not be complete, it can be still sound. Finally, investigating effective in-memory and on-disk data structures to speed up the resolution of the WSC problem can lead to new set of alternative solutions.

Several directions are ahead for future work. First, we will investigate the WSC problems with more expressive goals (e.g., goals specified in a temporal logic, LTL [24] and CTL [9]). Second, we plan to study efficient approximation solutions for the WSC problem.

## References

[1] J. L. Balcázar, J. Díaz, J. Gabarró, Structural Complexity 1, Springer-Verlag, 1988.
[2] J. L. Balcázar, J. Díaz, J. Gabarró, Structural Complexity 2, Springer-Verlag, 1990.
[3] B. Benatallah, M. Hacid, A. Léger, C. Rey, F. Toumani, On automating web services discovery, VLDB Journal 14 (1) (2005) 84–96.
[4] P. Bertoli, M. Pistore, Planning with extended goals and partial observability, in: Proceedings of the 14th Int'l Conf. on Automated Planning and Scheduling, ICAPS 2004, 2004, pp. 270–278.
[5] P. Bertoli, M. Pistore, P. Traverso, Automated web service composition by on-the-fly belief space search, in: Int'l Conf. on Automated Planning and Scheduling, ICAPS 2006, 2006, pp. 358–361.

[6] D. Beyer, A. Chakrabarti, T.A. Henzinger, Web service interfaces, in: Proceedings of the 14th Interational Conference on World Wide Web, WWW 2005, 2005, pp. 148–159.
[7] A. Chandra, D. Kozen, L. Stockmeyer, Alternation, Journal of the ACM 28 (1) (1981) 114–133.
[8] DARPA, DAML-S: Semantic markup for web services, 2003. http://www.daml.org/.
[9] E. Emerson, Temporal and modal logic, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, Elsevier Science Publishers, 1990, pp. 995–1072.
[10] W. Fan, F. Geerts, W. Gelade, F. Neven, A. Poggi, Complexity and composition of synthesized web services, in: Proceedings of the ACM Symposium on Principles of Database Systems, PODS 2008, 2008, pp. 231–240.
[11] X. Fu, T. Bultan, J. Su, Analysis of interacting bpel web services, in: Proceedings of the 13th International Conference on World Wide Web, WWW 2004, 2004, pp. 621–630.
[12] A. Herzig, J. Lang, D. Longin, T. Polacsek, A logic for planning under partial observability, in: Proceedings of National Conference on Artificial Intelligence, AAAI-00, 2000, pp. 768–773.
[13] R. Hull, J. Su, Tools for composite web services: a short overview, SIGMOD Record 34 (2) (2005) 86–95.
[14] R. Kazhamiakin, M. Pistore, L. Santuari, Analysis of communication models in web service compositions, in: Proceedings of Int'l Conf. on World Wide Web, WWW 2006, 2006, pp. 267–276.
[15] H. Kil, W. Nam, D. Lee, Computational complexity of web service composition based on behavioral descriptions, in: IEEE Int'l Conf. on Tools with Artificial Intelligence, 2008, pp. 359–363.
[16] M. Moffitt, On the partial observability of temporal uncertainty, in: Proceedings of National Conference on Artificial Intelligence, AAAI-07, 2007, pp. 1031–1037.
[17] M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg, S. Dustdar, A context-based mediation approach to compose semantic web services, ACM Transactions on Internet Technology 8 (1) (2007) 4.
[18] W. Nam, H. Kil, D. Lee, Type-aware web service composition using boolean satisfiability solver, in: Proceedings of IEEE Joint Conference on E-Commerce Technology, CEC'08, and Enterprise Computing, E-Commerce and E-Services, EEE'08, 2008, pp. 331–334.
[19] S. Narayanan, S.A. McIlraith, Simulation, verification and automated composition of web services, in: Proceedings of the 11th Int'l Conf. on World Wide Web, WWW 2002, 2002, pp. 77–88.
[20] OASIS, UDDI version 3.0.2, 2004. http://uddi.xml.org/.
[21] OASIS, Web services business process execution language version 2.0, 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.
[22] M. Pistore, P. Traverso, P. Bertoli, Automated composition of web services by planning in asynchronous domains, in: Proceedings of International Conference on Automated Planning and Scheduling, ICAPS 2005, 2005, pp. 2–11.
[23] M. Pistore, P. Traverso, P. Bertoli, A. Marconi, Automated synthesis of executable web service compositions from BPEL4WS processes, in: Special Interest Tracks and Posters of the 14th International Conference on World Wide Web, WWW 2005, 2005, pp. 1186–1187.
[24] A. Pnueli, The temporal semantics of concurrent programs, Theoretical Computer Science 13 (1981) 45–60.
[25] J. Rintanen, Complexity of planning with partial observability, in: Proceedings of International Conference on Automated Planning and Scheduling, ICAPS 2004, 2004, pp. 345–354.
[26] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, Journal of Computer and System Sciences 4 (2) (1970) 177–192.
[27] W3C, Web services activity, 2002. http://www.w3.org/2002/ws/.
[28] W3C, OWL-S: Semantic markup for web services, 2004. http://www.w3.org/Submission/OWL-S/.
[29] W3C, SOAP version 1.2, 2007. http://www.w3.org/TR/soap/.
[30] W3C, Web services description language (WSDL) version 2.0, 2007. http://www.w3.org/TR/wsdl20-primer/.
[31] Q. Yu, A. Bouguettaya, Framework for web service query algebra and optimization, ACM Transactions on Web 2 (1) (2008) 1–35.
[32] T. Yu, Y. Zhang, K.-J. Lin, Efficient algorithms for web services selection with end-to-end qos constraints, ACM Transactions on Web 1 (1) (2007) 6.