

---

# Behavioral Description Based Web Service Composition Using Abstraction and Refinement

---

**Hyunyoung Kil**

Korea Advanced Institute of Science & Technology, Daejeon 305-701, Korea  
E-mail: hkil@kaist.ac.kr

**Wonhong Nam\***

Konkuk University, Seoul 143-701, Korea  
E-mail: wnam@konkuk.ac.kr  
\*Corresponding author

**Dongwon Lee**

The Pennsylvania State University, University Park, PA 16802, USA  
E-mail: dongwon@psu.edu

**Abstract:** The *Web Service Composition (WSC)* problem with respect to behavioral descriptions deals with the automatic synthesis of a coordinator web service that controls a set of web services to reach a goal state. Despite its importance, however, solving the WSC problem for a general case (when the coordinator web service has only *partial observations*) remains *doubly exponential* in the number of variables in web service descriptions, rendering most attempts to compute an exact solution for modest size impractical. Toward this challenge, in this paper, we propose two novel (signature preserving and subsuming) approximation-based approaches using *abstraction* and *refinement*. Given a set of web service behavioral descriptions and a reachability goal, we automatically construct abstract web services which have less variables using *over-approximation*. If our method identifies a coordinator web service, the coordinator is guaranteed to control the given web services to reach the goal state no matter how they behave. Otherwise, our method refines the current abstraction by adding some variables that have strong dependency on the goal variables. We empirically validate that our proposals can solve realistic problems efficiently.

**Keywords:** Web Service Composition; Computational Complexity; Partial Observation; Behavioral Description; Abstraction; Refinement

**Reference** A preliminary version [21] of this paper was presented at the 21st International Joint Conference on Artificial Intelligence (IJCAI), pp. 1740–1745, 2009.

**Biographical notes:** Hyunyoung Kil received the B.S. and M.Sc. degrees from Korea University, Seoul, Korea, in 1998 and 2001, respectively. She received the M.S.E. degree from the University of Pennsylvania, Philadelphia, PA, USA in 2003, and the Ph.D. degree from the Pennsylvania State University, State College, PA, USA in 2010. She is currently a postdoctoral researcher of Korea Advanced Institute of Science and Technology, Daejeon, Korea. Her research interests include automated planning, web services composition, SOA and web science.

Wonhong Nam received the B.S. and M.Sc. degrees from Korea University, Seoul, Korea, in 1998 and in 2001, respectively, and the Ph.D. degree from the University of Pennsylvania, Philadelphia, PA, USA in 2007. From 2007 to 2009, he was a Postdoctoral Researcher with the College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, USA. He is currently an Assistant Professor of the Department of Internet & Multimedia Engineering, Konkuk University, Seoul, Korea. His research interests include formal methods, formal verification, model checking, automated planning, and web services composition.

Dongwon Lee received his B.E. degree in computer science from Korea University in 1993, his M.S. degree in computer science from Columbia University in 1995, and his Ph.D. degree in computer science from the University of California, Los Angeles in 2002. He is currently an associate professor in the College of Information Sciences and Technology, Pennsylvania State University. His research interests include databases, XML and Web analysis, and Web services and the Semantic Web.

---

## 1 Introduction

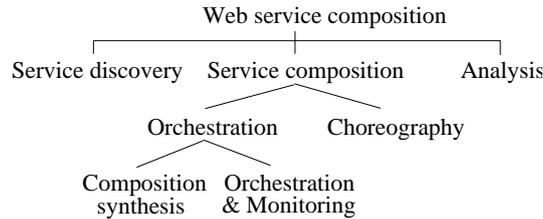
*Web services* are software systems designed to support machine to machine interoperation over the Internet. When a single web service does not satisfy a given requirement entirely, one needs to use a composition of web services. In particular, the *Web Service Composition (WSC)* problem that we focus on in this paper is defined as follows: *given a set of (behavioral descriptions of) web services,  $W$ , and a reachability goal,  $G$ , how to automatically synthesize a coordinator web service,  $c$ , that controls  $W$  to satisfy  $G$ .* In this paper, a behavioral description of a web service is a formal specification on how the web service executes internally and externally by interacting with users; e.g., describing what output value it returns for a given input and its state, and how it changes its internal state.

Despite abundant research on the WSC problem, only a few (e.g., [43, 32, 31]) employ realistic models with partial observation. Our previous work [20] showed that: (1) the WSC problem for a restricted case (when the synthesized coordinator web service,  $c$ , has *full observation* for all variables) is EXP-hard, and (2) the WSC problem for a general case (when  $c$  has *partial observation*) is 2-EXP-hard. These results imply that more focus should be put on the study of approximate solutions to the WSC problem. Toward this challenge, in this paper, we propose two approximation-based algorithms using “abstraction and refinement” [12]. To the best of our knowledge, it is the first attempt to apply an abstraction technique to the WSC problem. Even, in *planning under partial observation* [7, 37] which has a strong connection to the WSC problem, no study has attempted to apply abstraction techniques.

The first step is to reduce the original web services to the abstract ones with less variables. If we identify a coordinator that controls the abstract web services to satisfy a given goal, the coordinator can control the original web services to satisfy the goal since the abstract web services “over-approximate” the concrete ones. Otherwise, we refine the abstract web services by adding variables, and repeat to find a solution. For the abstraction, we propose two methods—*signature-preserving abstraction* and *signature-subsuming abstraction*. Finally, we report the performance of our tool using three sets of realistic problems (i.e., 8 instances), comparing it with a basic algorithm [43] without abstraction/refinement. Our experiment shows that our proposal outperforms the basic algorithm. Finally, it is worth pointing out that our approach can be readily adopted for other WSC techniques such as knowledge-level composition [31].

## 2 Background

The web service composition (WSC) is required when a client request cannot be fulfilled by a single pre-existing web service. In such a case, one prefers integrating existing web services to satisfy the request. In the work [1], this new value-added service and a process to generate the service are named as a *composite service* and *composition*, respectively. The essential activities of the WSC problem are illustrated like Figure 1.



**Figure 1** The activities of web service composition problem.

At first, *service discovery* is to find out one or a set of web service appropriate to a given request. How to locate a web service is also an inseparable issue in the discovery. For this, many researchers have focused on a centralized UDDI registry [28] as an effective method to solve a web service discovery problem. UDDI [28], a standard for centralized repositories, stores information describing web services produced by service providers, and is queried by service requesters. As an alternative of UDDI, specialized portals, such as XMethods [45], BindingPoint [8], RemoteMethods [36], or eSynaps [14] appear. They gather web services via the manual registration and support a keyword-based service search by focused crawlers. However, as the number of web services grows, such a centralized approach quickly becomes impractical. As a result, systems building on ontology and/or using Peer-to-Peer (P2P) technologies have been introduced (e.g., Hypercube ontology-based P2P system [38] and the Speed-R [41]).

Next, *service composition* integrates the existing services for an given request. Since various research fields tackle the composition problem with their own flavors, the vocabulary representing the design issues in the composition is mixed in many literatures. Here, we identify these terms for general understanding of the composition issues:

- *Orchestration vs. choreography*: Both of orchestration and choreography describe how web services can interact at the message level, including the business logic and execution order of the interactions. However, orchestration methods aim at synthesizing a new web service called a *coordinator* which has a specialized role of controlling the other services by properly exchanging messages. WS-BPEL [29] is an example of a standard language for orchestration. A choreography, on the other hand, is more collaborative in nature where each party involved in the process describes its own role for one shared goal. Accordingly, the execution of the composition is distributed to all participating web services. As a result, a specification generated by the orchestration identifies the execution steps for the participating services while a choreography specification describes the set of allowable conversation for a composite web service. Their difference in the topology of the composite service clearly explains why orchestration methods are called centralized or mediated-based approaches whereas choreography methods are called distributed or peer-to-peer.
- *Composition synthesis vs. orchestration*: Orchestration is frequently used with a limited meaning by separating a composition synthesis in literatures [16, 1, 19, 5]. Composition synthesis concerns how to generate a specification of

how to coordinate the participating services, whereas orchestration here takes into account how to coordinate the various participating services by executing the specification generated by the composition synthesis. It also includes the functionality to monitor control and data-flow among the participants for the correct execution of the composite service.

Note that in this paper, we deal with the composition synthesis. However, the service composition is more generally used as a representative, we use the term, *web service composition problem*, instead of composition synthesis.

In addition, there are other issues that give an influence to the design of a composite service. Both *static* and *dynamic* composition present the clear difference in service selection timing. In the static composition method, we can decide the services to be composed at design time. On the other hand, in the dynamic composition, it can happen at run time.

Finally, *analysis (verification)* in Figure 1 is particularly necessary since, by using automatic algorithms, a composite service is to be created from pre-existing services. The ultimate goal is to ensure that the eventual execution of a composite service produces the desired behavior. Ideally, one would be able to statically verify properties (e.g., in a temporal logic) for composite services. There have been various attempts at developing such analysis methods for web services and workflow systems.

### 3 Related Work

On the WSC problem, extensive research (e.g., [23, 39, 43, 32, 31, 26]) has been carried out. In this section, we review some of the representative works.

First, automated planning techniques are mainly applied to solve the WSC problem due to its similarity to the planning problem. The work [23] presents a composition method to use logical inferencing techniques where a service capability specified in DAML-S is translated to situation calculus and Golog. Golog is designed for the specification and execution of complex actions. Hierarchical Task Network (HTN) planning [39] allows the expression of aggregate behaviors. As an input, a planner takes composite processes representing how to compose a sequence of single step actions and tries to find a collection of atomic process instances which form an execution path for top-level composite process. However, these works consider neither non-determinism nor partial observability.

In [43, 32, 31], Pistore et al. have defined web service compositions over a planning domain where uncertainty on a state and a non-deterministic action are allowed, and presented algorithms which take into account service behaviors and complex goals. They extend the previous work in order to handle asynchronous, message-based interaction between domains and plans [32]. However, they do not consider the complexity of the WSC problem. Recently, Fan et al. [15] investigate the complexity of the WSC based on query rewriting using views, but do not include a more realistic non-deterministic web service in their setting. Furthermore, even though some work considered planning with abstraction, there has been no work that applied an abstraction technique to the WSC problem. Huang et al. [18] propose a method to reduce observation variables for strong plans. This technique,

however, cannot identify such variables until a plan is fully constructed. Armano et al. [3] employ abstraction techniques for a hierarchical planner, and Smith et al. [40] present an abstraction technique to generate exponentially smaller POMDP. Therefore, their techniques cannot be applied to our setting.

Recently, some works [9, 24, 46] focus on service process behaviors for fully automatic composition. Brogi [9] for instance discusses the requirement of service behavior representation in web services and the potential advantages of exploiting this behavioral information for service discovery and composition. The Adaptive Services Grid (ASG) project [24] presents how dynamic adaptation strategies can support an automated selection, composition and binding of services during runtime. Yu et al. [46] synthesize the process model from automata of the rules, by using pathfinding, branching structure identification and parallel structure identification.

In addition to the aforementioned research, our work is also related with the following; e.g., game theory [35, 42], open system synthesis with temporal logics [2, 34, 44, 22] and automated planning [17, 7, 37, 25].

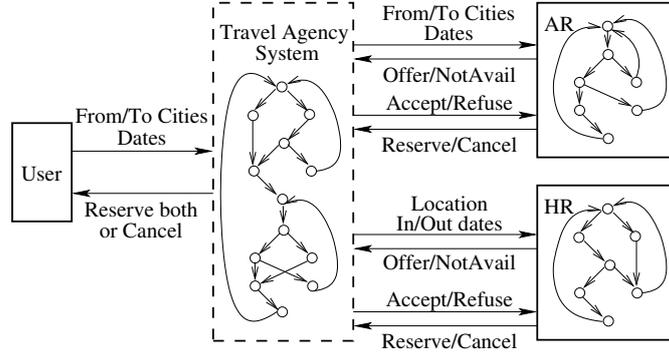
First, the WSC problem has a close connection to game theory. Since the coordinator cannot control the non-determinism of a given set of web services and has to decide inputs to web services only with outputs from them, this problem can be considered as a two-player game with partial observation where the coordinator wants to win over the web services. Reif [35] has proved that the problem of determining the outcome of universal games from a given initial position is 2-EXP-hard. Thomas [42] has surveyed the algorithmic theory of infinite games, and its role in automatic program synthesis and verification.

For open systems, next, various controller synthesis problems have been studied with temporal logics [33, 13]. Most have considered linear time logics [33] and often involved dealing with incomplete information due to partial observation [34, 44]. For branching-time cases, Antoniotti [2] has studied the synthesis of memoryless controllers with maximal environments. Kupferman and Vardi [22] have studied the open system synthesis under partial observation, and proved that the problem with a specification in CTL (CTL\*) [13] is EXP-complete (2-EXP-complete, respectively).

Finally, the WSC problem is related to automated planning under partial observation [17, 7, 37, 25]. Herzig et al. [17] have proposed a dynamic logic EDL for planning under partial observability. In [7], 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 [37]. Moffitt [25] has explored a means to both model and reason about partial observability within the scope of constraint based temporal reasoning.

## 4 Web Service Composition

**Example 1 (Travel agency system)** *Clients want to reserve both a flight ticket and a hotel room for a particular destination and a period. However, there exist only an airline reservation (AR) web service and a hotel reservation (HR) web service separately. Clearly, we want to combine these web services rather than implementing a new one. One way to combine them is to automatically construct a coordinator web service which communicates with them to book both a flight ticket*



**Figure 2** Travel agency system

and a hotel room. Figure 2 illustrates this example. The 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, the HR service is requested with check-in/check-out dates and a location, and then checks the number of available rooms. If there is available accommodation, 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 both a flight ticket and a hotel room 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 run-time 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 obviously be represented by a deterministic state-transition system.

**Definition 1 (Web service)** A (behavioral description of 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. A state  $s$  of  $w$  is a valuation for every variable in  $X$ . We denote a set of all states as  $S$ .
- $X^I$  is a finite set of input variables that  $w$  reads from the outside;  $X \cap X^I = \emptyset$ , and every variable  $x \in X \cup X^I$  has a finite domain (e.g., Boolean, bounded integers, or enumerated types). A state 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 other web services can read. Let us denote a set of input and output variables by  $X^{IO}$  (i.e.,  $X^{IO} = X^I \cup X^O$ ), and a set of all variables by  $X^A$  (i.e.,  $X^A = X \cup X^I$ ).

- $Init(X)$  is an initial predicate over  $X$ .  $Init(s) = true$  if and only if  $s$  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  $X' = \{x' \mid x' \text{ for each } x \in X\}$ , which represents a set of variables encoding successor states.  $T(s, in, s')$  is true if and only if  $s'$  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.

While the formalism for web services by Traverso et al. [43] is based on an explicit state-transition system using a set of states, we *symbolically* define web services by a set of variables, which is more compact.

**Example 2** Consider a simple version of a web service  $w$  for the airline reservation in Example 1, and assume that clients can request (reserve or refuse) a flight ticket by an action  $req_1$  or  $req_2$  (accept or refuse, respectively). The web service  $w$  can be represented as  $(X, X^I, X^O, Init, T)$  where:

- $X = \{\text{state}, \text{avail}, \text{reply}, \text{confirm}, \text{f\_num}, \text{tr\_num}\}$  where  $\text{state}$  has the domain  $\{q_1, q_2\}$ ,  $\text{avail}$  is boolean,  $\text{reply}$  has the domain  $\{\text{undecided}, \text{offer}, \text{notAvail}\}$ ,  $\text{confirm}$  has the domain  $\{\text{undecided}, \text{reserve}, \text{cancel}\}$ ,  $\text{f\_num}$  (flight number) has the domain  $\{f_1, f_2\}$ , and  $\text{tr\_num}$  (transaction number) has the domain  $\{t_1, t_2\}$ .
- $X^I = \{\text{action}\}$  where  $\text{action}$  has the domain  $\{req_1, req_2, \text{accept}, \text{refuse}\}$ .
- $X^O = \{\text{reply}, \text{confirm}, \text{f\_num}\}$ .
- $Init(X) \equiv (\text{state} = q_1) \wedge (\text{reply} = \text{undecided}) \wedge (\text{confirm} = \text{undecided})$ .
- $T(X, X^I, X') \equiv$   
 $((\text{state} = q_1) \wedge (\text{action} = req_1) \wedge (\text{avail} = true)) \rightarrow$   
 $((\text{state}' = q_2) \wedge (\text{reply}' = \text{offer}) \wedge (\text{tr\_num}' = t_1))$   
 $\wedge (((\text{state} = q_1) \wedge (\text{action} = req_1)) \rightarrow (\text{f\_num}' = f_1))$   
 $\wedge \dots$   
 $\wedge (((\text{state} = q_2) \wedge (\text{action} = \text{accept})) \rightarrow$   
 $((\text{state}' = q_1) \wedge (\text{confirm}' = \text{reserve})))$   
 $\wedge (((\text{state} = q_2) \wedge (\text{action} = \text{refuse})) \rightarrow$   
 $((\text{state}' = q_1) \wedge (\text{confirm}' = \text{cancel}))).$

Note that the process model for any web service described in semantic web languages (e.g., WS-BPEL or OWL-S) can be easily transformed into our representation above without any information loss if it has only finite domain variables and no recursion<sup>1</sup>. In the WSC problem in this paper, given a set of available web services,  $W$ , every web service in  $W$  communicates only with their coordinator but not with each other.

**Definition 2 (Set of web services)** Based on the assumption above, given a set  $W = \{w_1, \dots, w_n\}$  of web services where for each  $w_i(X_i, X_i^I, X_i^O, \text{Init}_i, T_i)$ ,  $X_i$  is disjoint with every other  $X_j$  and  $X_i^I$  is disjoint with every other  $X_j^I$ ,  $W$  also can be represented by a 5-tuple  $(X, X^I, X^O, \text{Init}, T)$  where  $X = X_1 \cup \dots \cup X_n$ ,  $X^I = X_1^I \cup \dots \cup X_n^I$ ,  $X^O = X_1^O \cup \dots \cup X_n^O$ ,  $\text{Init}(X) = \text{Init}_1 \wedge \dots \wedge \text{Init}_n$ , and  $T(X, X^I, X^O) = T_1 \wedge \dots \wedge T_n$ .

In the above definition, we assume that web services in  $W$  do not share the identical variable name. If this case happens, by simply adding unique prefix/suffix to the names, we can distinguish the variables.

Since a *coordinator web service* is also a web service, it is a 5-tuple  $c(X_c, X_c^I, X_c^O, \text{Init}_c, T_c)$ . In what follows,  $s_c$  denotes a state of a coordinator web service, and  $S_c$  denotes a set of all states of a coordinator. 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  $s_c$  and input  $in$ , there exists only one next coordinator state  $s'_c$  such that  $T_c(s_c, in, s'_c) = \text{true}$ .

**Example 3** Consider a simple coordinator web service  $c$  that communicates with AR in Example 2. The coordinator web service  $c = (X_c, X_c^I, X_c^O, \text{Init}_c, T_c)$  can be represented with the following elements:

- $X_c = \{\text{c\_state}, \text{action}\}$  where  $\text{c\_state}$  has the domain  $\{s_1, s_2\}$ , and  $\text{action}$  has the domain  $\{\text{req}_1, \text{req}_2, \text{accept}, \text{refuse}\}$ .
- $X_c^I = \{\text{reply}, \text{confirm}, \text{f\_num}\}$  where  $\text{reply}$  has the domain  $\{\text{undecided}, \text{offer}, \text{notAvailable}\}$ ,  $\text{confirm}$  has the domain  $\{\text{undecided}, \text{reserve}, \text{cancel}\}$ , and  $\text{f\_num}$  has the domain  $\{f_1, f_2\}$ .
- $X_c^O = \{\text{action}\}$ .
- $\text{Init}_c(X_c) \equiv (\text{c\_state} = s_1) \wedge (\text{action} = \text{req}_1)$ .
- $T_c(X_c, X_c^I, X_c^O) \equiv$   
 $((\text{c\_state} = s_1) \wedge (\text{reply} = \text{offer})) \rightarrow$   
 $((\text{c\_state}' = s_2) \wedge (\text{action}' = \text{accept}))$   
 $\wedge (((\text{c\_state} = s_1) \wedge (\text{reply} = \text{notAvailable})) \rightarrow$   
 $((\text{c\_state}' = s_1) \wedge (\text{action}' = \text{req}_2)))$   
 $\wedge (((\text{c\_state} = s_2) \wedge (\text{confirm} = \text{reserve})) \rightarrow$   
 $((\text{c\_state}' = s_1) \wedge (\text{action}' = \text{req}_1)))$   
 $\wedge (((\text{c\_state} = s_2) \wedge (\text{confirm} = \text{cancel})) \rightarrow$   
 $((\text{c\_state}' = s_1) \wedge (\text{action}' = \text{req}_1))).$

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  $Y$ .

**Definition 3 (Execution tree)** Given a set  $W(X, X^I, X^O, \text{Init}, T)$  of web services and a coordinator  $c(X_c, X_c^I, X_c^O, \text{Init}_c, T_c)$  where  $X^I = X_c^I$  and  $X^O = X_c^O$ , 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  $Init_c(s_c) = true$ .
- Each node  $(s, s_c)$  has a set of child nodes,  $\{(s', s'_c) \mid T(s, in, s')=true, in=s_c[X^I], T_c(s_c, in_c, s'_c)=true, in_c=s'[X^O]\}$ .

In the above, intuitively, the web services  $W$ , by receiving the input  $in$  from the current state  $s_c$  of the coordinator, collectively proceed from  $s$  to the next state  $s'$ , and then the coordinator, by receiving the input  $in_c$  from the new state  $s'$  of the web services, proceeds from  $s_c$  to the next state  $s'_c$ . Even though the composition of  $W$  and  $c$  is defined as synchronous communication, we can easily extend this model for *asynchronous* communication using  $\tau$ -transition [32]. A goal  $G \subseteq S$  is a set of states to reach, and specified as a predicate.

**Definition 4 (Web service composition problem)** *Given a set  $W$  of web services, a coordinator  $c$ , and a goal  $G$ , we define  $W||c \models G$  if for every path  $(s^0, s_c^0)(s^1, s_c^1) \dots$  in the execution tree  $W||c$ , there exists  $i \geq 0$  such that  $s^i \in G$ ; namely, every path from the initial node  $(s^0, s_c^0)$  reaches a goal state eventually. The 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  $W||c \models G$ .*

**Example 4** *In Example 1, we wish to reserve both a flight ticket and a hotel room. This can be represented as  $G \equiv (\text{flightConfirm} = \text{reserve}) \wedge (\text{hotelConfirm} = \text{reserve})$ . Now, given a set  $W = \{w_{AR}, w_{HR}\}$  of web services and the goal  $G$  above, a WSC problem is to construct a coordinator web service  $c$  such that  $W||c \models G$ .*

To study the computational complexity (i.e., lower bound) for WSC, we define two WSC problems as follows:

- **WSC with full observation:** a special case of WSC problems where  $W(X, X^I, X^O, Init, T)$  such that  $X = X^O$ ; i.e.,  $W$  contains no internal variable.
- **WSC with partial observation:** a general WSC problem where there is no restriction for  $X^O$ . That is, a coordinator can read only the output variables in  $X^O$ .

## 5 Lower Bounds

In this section, we study the computational complexities (lower bounds) for the two decision problems we defined in Section 4. Our proofs use reductions from an alternating Turing machine (ATM) [30], which show the space complexities for the problems. Finally, based on Theorem 1, we establish the time complexities. While our previous work [27] has presented the tight bounds (i.e., lower bounds and upper bounds) of more various problem settings, this paper focuses on presenting an efficient approximation-based algorithm based on the complexity result.

### 5.1 Definition

An alternating Turing machine (ATM) [30] is a tuple  $A = (Q, \Sigma, q_0, \delta, l)$  where

- $Q$  is a finite set of states,  $\Sigma$  is a finite tape alphabet, and  $q_0 \in Q$  is the initial state.
- $\delta : Q \times \Sigma \rightarrow 2^{Q \times \Sigma \cup \{\#\} \times \{\mathcal{L}, \mathcal{N}, \mathcal{R}\}}$  is a transition function where  $\{\mathcal{L}, \mathcal{N}, \mathcal{R}\}$  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  $A(Q, \Sigma, q_0, \delta, l)$  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 [30].

### 5.2 Computational complexity

In this paper, we consider the following complexity classes [30].  $\text{DTIME}(f)$  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.

$$\begin{aligned}
 \text{P} &= \bigcup_{k \geq 0} \text{DTIME}(n^k) \\
 \text{EXP} &= \bigcup_{k \geq 0} \text{DTIME}(2^{n^k}) \\
 \text{2-EXP} &= \bigcup_{k \geq 0} \text{DTIME}(2^{2^{n^k}}) \\
 \text{APSPACE} &= \bigcup_{k \geq 0} \text{ASPACE}(n^k) \\
 \text{AEXPSPACE} &= \bigcup_{k \geq 0} \text{ASPACE}(2^{n^k})
 \end{aligned}$$

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. [10] have proved interesting connections between time complexities on deterministic Turing machines and space complexities on ATMs:

**Theorem 1** ([10])  $\text{EXP} = \text{APSPACE}$ , and  $\text{2-EXP} = \text{AEXPSPACE}$ .

5.3 *WSC with full observation*

**Theorem 2** *The WSC problem with full observation is EXP-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  $\sigma$  if and only if there exists a coordinator to satisfy a goal. We prove it using the following lemmas.

**Lemma 1** *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  $W$  and a goal  $G$ .*

**Proof 1** *We can construct a set  $W(X, X^I, X^O, \text{Init}, T)$  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  $1 \leq i \leq p(n)$ , each variable  $cl_i$  is the contents of the  $i$ th tape cell; its domain is  $\Sigma \cup \{\#\}$ .*
- *hd describes the R/W head position; its domain is  $\{1, \dots, p(n)+1\}$ .*
- *label represents the label of the current state; it has the domain,  $\{\forall, \exists, \text{accept}\}$ .*

*The set of input variables is  $X^I = \{\text{input}\}$  where the domain of input is  $\{A_{(q,i,a)} \mid q \in Q, l(q) = \forall, 0 \leq i \leq p(n), a \in \Sigma\} \cup \{E_{(q,i,a,j)} \mid q \in Q, l(q) = \exists, 0 \leq i \leq p(n), a \in \Sigma, 0 \leq j \leq |\delta(a, q)|\}$ . The set  $X^O$  of output variables equals to  $X$  since this problem is based on the full observation. As the initial configuration of  $A$ , the initial state predicate  $\text{Init}(X)$  is  $(\text{state} = q_0) \wedge \bigwedge_{1 \leq i \leq n} (cl_i = a_i) \wedge \bigwedge_{n < i \leq p(n)} (cl_i = \#) \wedge (\text{hd} = 1) \wedge (\text{label} = l(q_0))$ . Note that the input string is  $\sigma = a_1 \cdots a_n$ . The transition predicate  $T(X, X^I, X^O)$  is  $((\text{hd} = p(n)+1) \rightarrow T_V) \wedge ((\text{label} = \forall) \rightarrow T_\forall) \wedge ((\text{label} = \exists) \rightarrow T_\exists)$  with the following sub-formulae*

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

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

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

**Lemma 2** *If  $\sigma \in L(A)$ , then there exists a coordinator  $c = (X_c, X_c^I, X_c^O, \text{Init}_c, T_c)$  such that  $W||c \models G$ .*

**Proof 2** *As per Section 5.1,  $\sigma \in L(A)$  means that the initial configuration of  $A$  with respect to  $\sigma$  is  $m$ -accepting ( $m \geq 0$ ). Now, we show that there exists a coordinator  $c$  such that for every path  $(s^0, s_c^0)(s^1, s_c^1) \dots$  in the execution tree  $W||c$ , there exists  $s^i \in G$ . The coordinator to be constructed is  $c = (X_c, X_c^I, X_c^O, \text{Init}_c, T_c)$  where  $X_c = \{\text{input}\}$ ,  $X_c^I = X$ , and  $X_c^O = \{\text{input}\}$ . We will provide  $\text{Init}_c$  and  $T_c$  in the later of this proof.*

*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  $m$ -accepting, each successor is at most  $(m-1)$ -accepting.*
- *For each  $cf = (q, \sigma_1, \sigma_2) \in \Upsilon$  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  $\sigma$ . 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  $cf$  in  $ACT$  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*

- $s(\text{state}) = q$ .
- *For  $1 \leq i \leq |\sigma|$ ,  $s(\text{cl}_i) = \sigma[i]$  where  $\sigma = \sigma_1\sigma_2$ , and for  $|\sigma| < i \leq p(n)$ ,  $s(\text{cl}_i) = \#$ .*
- $s(\text{hd}) = |\sigma_1|$ .
- $s(\text{label}) = l(q)$ .

*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  $s_c(\text{input}) = A_{(q,i,a)}$  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  $\delta(q, a) = \{(q_1, a_1, m_1), \dots, (q_k, a_k, m_k)\}$  where  $a = \sigma_1[|\sigma_1|]$ . Now, if  $l(q) = \exists$ ,  $s_c(\text{input}) = E_{(q,i,a,j)}$  where  $i = |\sigma_1|$  and  $a = \sigma_1[i]$ .*

*According to  $\alpha$  and  $\beta$ , we have an execution tree of  $W||c$  where each node is  $(\alpha(cf), \beta(cf))$ . Now, by induction, we can show 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. That is, if  $\sigma \in L(A)$ , there exists a coordinator  $c = (X_c, X_c^I, X_c^O, \text{Init}_c, T_c)$  such that  $W||c \models G$ . For the detail of the induction, see [20].*

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

**Proof 3** *As per Section 4, the fact that there exists a coordinator  $c$  such that  $W||c \models G$  means that every path  $(s^0, s_c^0)(s^1, s_c^1) \dots$  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  $(s^0, s_c^0)(s^1, s_c^1) \dots$  of  $W||c$ , its prefix ending at a goal state (i.e.  $(s^0, s_c^0) \dots (s^k, s_c^k)$  such that  $s^k \in G$ ). In what follows, 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  $s(\text{state})=q$ ,  $s(\text{cl}_i)=b_i$  where  $1 \leq i \leq p(n)$ ,  $s(\text{hd})=i$  and  $s(\text{label})=l(q)$ , we have a corresponding configuration  $cf = \gamma(s) = (q, \sigma_1, \sigma_2)$  such that  $\sigma_1 = b_1 \dots b_i$  and  $\sigma_2 = b_{i+1} \dots b_{k-1}$  where  $k$  is the index of the first appearance of  $\#$ .*

*Now, by induction, we can show 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. That is, if there exists a coordinator  $c$  such that  $W||c \models G$ , then  $\sigma \in L(A)$ .*

#### 5.4 WSC with partial observation

**Theorem 3** *The WSC problem with partial observation is 2-EXP-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  $A = (Q, \Sigma, q_0, \delta, l)$  with exponential space bound  $e(n)$  and an input string  $\sigma = a_1 \dots a_n$  (where  $n = |\sigma|$ ), we can construct a WSC problem instance with  $W$  and a goal  $G$ .*

**Proof 4** *An important difference with full observation problem 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  $cl_i$ , 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  $W(X, X^I, X^O, \text{Init}, T)$  of web services 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  $\{1, \dots, e(n)\}$ .*
- *cl represents the contents of the cell of which index is  $idx$ ; its domain is  $\Sigma \cup \{\#\}$ .*
- *hd; its domain is  $\{1, \dots, e(n)+1\}$ . For  $idx$  and  $hd$ , we need only  $\lceil \log_2(e(n)+1) \rceil$  bits.*

- *label*; it has a domain,  $\{\forall, \exists, \text{accept}\}$ .
- *lsb* represents the symbol written by the head in the last step; it has a domain,  $\Sigma \cup \{\#\}$ .

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

- $T_V \equiv (\text{state}' = \text{state}) \wedge (\text{hd}' = \text{hd}) \wedge (\text{label}' = \text{label})$
- $T_\forall \equiv \bigwedge_{q \in Q, a \in \Sigma} (((\text{state} = q) \wedge ((\text{hd} = \text{idx}) \rightarrow (\text{cl} = a)) \wedge (\text{input} = A_{(q,a)})) \rightarrow \bigvee_{1 \leq j \leq k} ((\text{hd} = \text{idx}) \rightarrow ((\text{state}' = q_j) \wedge (\text{cl}' = a_j) \wedge (\text{hd}' = \text{hd} + \Delta) \wedge (\text{label}' = l(q_j)) \wedge (\text{lsb}' = a_j))))$
- $T_\exists \equiv \bigwedge_{q \in Q, a \in \Sigma, 1 \leq j \leq k} (((\text{state} = q) \wedge ((\text{hd} = \text{idx}) \rightarrow (\text{cl} = a)) \wedge (\text{input} = E_{(a,q,j)})) \rightarrow (((\text{hd} = \text{idx}) \rightarrow (\text{state}' = q_j) \wedge (\text{cl}' = a_j) \wedge (\text{hd}' = \text{hd} + \Delta) \wedge (\text{label}' = l(q_j)) \wedge (\text{lsb}' = a_j))))$

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

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

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

**Proof 5** *Given  $A$  such that  $\sigma \in L(A)$ , we can construct a coordinator  $c = (X_c, X_c^I, X_c^O, \text{Init}_c, T_c)$  where  $X_c = \{\text{input}\}$ ,  $X_c^I = \{\text{state}, \text{lsb}\}$ , and  $X_c^O = \{\text{input}\}$ . As 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  $\bigwedge_{q \in Q, a \in \Sigma} (((\text{state} = q) \wedge (\text{cl} = a)) \rightarrow (\text{input}' = A_{(q,a)}))$ . Otherwise,  $\bigwedge_{q \in Q, a \in \Sigma} (((\text{state} = q) \wedge (\text{cl} = a)) \rightarrow (\text{input}' = E_{(q,a,j)}))$  where  $j$  is the index of the transition by which the ATM proceeds from the corresponding  $\exists$ -configuration to the next in  $\text{ACT}_{(A,\sigma)}$ . Similarly with  $T_c$ , we can define the initial predicate  $\text{Init}_c$  as  $((l(q_0) = \forall) \rightarrow (\text{input} = A_{(q_0, a_1)})) \wedge ((l(q_0) = \exists) \rightarrow (\text{input} = E_{(q_0, a_1, j)}))$  where  $a_1$  is the first symbol of the input string  $\sigma$  and  $j$  is obtained as the above.*

Now, we show that  $\text{ACT}_{(A,\sigma)}$  is mapped into an execution tree  $W \parallel c$ . For this mapping, we have two mapping functions,  $\alpha$  and  $\beta$ ;  $\alpha$  maps a configuration  $cf$  in  $\text{ACT}$  to a state  $s$  of web services  $W$ , and  $\beta$  maps  $cf$  to a state  $s_c$  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

- $s(\text{state}) = q$ .
- $s(\text{cl}) = \sigma[i]$  if  $i \leq \sigma$  where  $\sigma = \sigma_1 \sigma_2$ ; otherwise,  $s(\text{cl}) = \#$ .

- $s(idx) = i$ .
- $s(hd) = |\sigma_1|$ .
- $s(label) = l(q)$ .

The mapping function  $\beta$  is the same with  $\beta$  in Lemma 2. Now, we claim that if  $cf$  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.

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 6** *For the finite subtree  $ST$  of  $W||c$ , we construct  $ACT_{(A, \sigma)}$ . 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  $s(state)=q$ ,  $s(cl)=a_1$ ,  $s(idx)=i$ ,  $s(hd)=h$ ,  $s(label)=l(q)$ , and  $s(lsb)=a_2$ , 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  $(\sigma_2\sigma'_2)[|\sigma_1|] = a_2$ .*

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.

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.

## 6 Basic Algorithm for WSC Problem

In this section, we study a basic algorithm for the general WSC problem defined in Section 4. Several researches [43, 32] have successfully applied a planning technique with partial observation [6] to WSC problems. Thus, we also employ the same method for our baseline algorithm; Algorithm 1 for the WSC problem is based on the automated planning algorithm on partial observation [6]. In a general case of WSC, a coordinator web service is not able to identify the exact state of target web services due to partial observation. Hence, we model this uncertainty by using a *belief state*, which is a set of *possible* states of target web services but *indistinguishable*. For example, given two Boolean variables,  $x$  and  $y$ , assume that  $y$  is not observed by a coordinator. In this case, although 4 states may actually exist, the coordinator can distinguish only two states based on  $x$ , which are called “belief states.” The underlying idea of Algorithm 1 is to construct an *and-or searching tree*

**Algorithm 1:** WSC with partial observation**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 ( $States(Init) \subseteq States(G)$ ) 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  $null;$ 

```

---

from initial belief states to goal belief states. That is, from any node (a belief state) of the tree, for non-determinism of output values of web services, we extend the tree with a set of child nodes via *and-edges*. In this case, all the child nodes should reach a goal belief state. For the coordinator selecting input values, we construct a set of child nodes via *or-edges*. In this case, at least one child is required to reach a goal belief state.

To initialize the and-or searching tree, Algorithm 1 first constructs a root node (a belief 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<sup>2</sup> the tree from the selected node by computing a set of possible successor nodes (line 8), and (3) check if the node can reach a goal state based on the and-or constraint (lines 9–14). Once we identify the result of each node, we propagate the result to its ancestor node (line 15). Finally, if the algorithm identifies the result of root node as *true*, it constructs a coordinator web service from the tree, and returns the coordinator (lines 17–19). Otherwise, it returns *null* (line 20). The complexity of the algorithm is  $O(2^{2^n})$  where  $n$  is the number of variables in  $W$ , since the number of states of  $W$  is  $2^n$  and thus the number of belief states is  $2^{2^n}$  (recall Theorem 3).

## 7 Signature-preserving Abstraction and Refinement

Theorems 2 and 3 imply that the WSC problem is computationally hard. Hence, more efforts to devise efficient approximation solutions to the WSC problem are needed. In addition, the complexity of Algorithm 1 also provides the same implication. Therefore, we propose two approximation-based methods using abstraction and refinement in Sections 7 and 8.

### 7.1 Signature-preserving abstraction

Given a set  $W$  of web services, we define *signature-preserving abstract web services* that have the same signature (i.e., the same I/O variables) but less variables than  $W$ .

**Definition 5 (Signature-preserving abstract web services)** *Given a set of web services  $W(X, X^I, X^O, \text{Init}, T)$  and a set  $Y$  of variables such that  $X^{IO} \subseteq Y \subseteq X^A$ , the signature-preserving abstraction of  $W$  with respect to  $Y$  is  $W_Y(X_Y, X_Y^I, X_Y^O, \text{Init}_Y, T_Y)$  where:*

- $X_Y = Y \setminus X^I$ ,  $X_Y^I = X^I$ , and  $X_Y^O = X^O$ .
- For every  $s_Y \in S_Y$ ,  $\text{Init}_Y(s_Y) = \text{true}$  iff  $\exists s \in S. (\text{Init}(s) = \text{true}) \wedge (s_Y = s[X_Y])$ .
- For every  $s_Y, s'_Y \in S_Y$ ,  $T_Y(s_Y, \text{in}, s'_Y) = \text{true}$  iff  $\exists s, s' \in S. (T(s, \text{in}, s') = \text{true}) \wedge (s_Y = s[X_Y]) \wedge (s'_Y = s'[X_Y])$ .

Since  $W_Y$  preserves the signature of  $W$ , once we construct a coordinator  $c$  which can be composed with  $W_Y$  based on Definition 3,  $c$  also can be composed with  $W$ . Moreover, since the abstraction  $W_Y$  over-approximates the concrete web services  $W$  (i.e.,  $W_Y$  contains all the behaviors of  $W$ ),  $W_Y$  satisfies the following property.

**Theorem 4 (Soundness)** *Given a set  $W$  of web services and a goal  $G$ , if a coordinator web service  $c$  satisfies  $W' || c \models G$  where  $W'$  is a signature-preserving abstraction of  $W$  (e.g.,  $W_Y$  in Definition 5), then  $c$  also satisfies  $W || c \models G$ .*

**Example 5 (Abstraction)** *Figure 3(a) illustrates the concrete state space with 6 states, where there are three internal variables—`state`, `avail`, `tr_num`. Symbols above arrows represent a value of an input variable. In this example, from the state  $s_1$ , we have a strategy to guarantee to reach GOAL—invoking `req` and `order`. Figure 3(b) shows an abstract state space with respect to  $\{\text{state}, \text{avail}\}$ .  $s_1$  and  $s_4$  in the original space are mapped to  $s_7$  and  $s_9$ , respectively. Two states,  $s_2$  and  $s_3$ , ( $s_5$  and  $s_6$ ) collapse into  $s_8$  ( $s_{10}$ , respectively). Although the number of states decreases, every path in the original state space is mapped to one of the paths in the abstract space. Moreover, from the state  $s_7$  corresponding to  $s_1$ , we still have a strategy to guarantee to reach GOAL. Figure 3(c) shows a coarser abstraction. However, from the state  $s_{11}$  corresponding to  $s_1$ , we no longer have a strategy to guarantee to reach GOAL since we abstract out too much.*

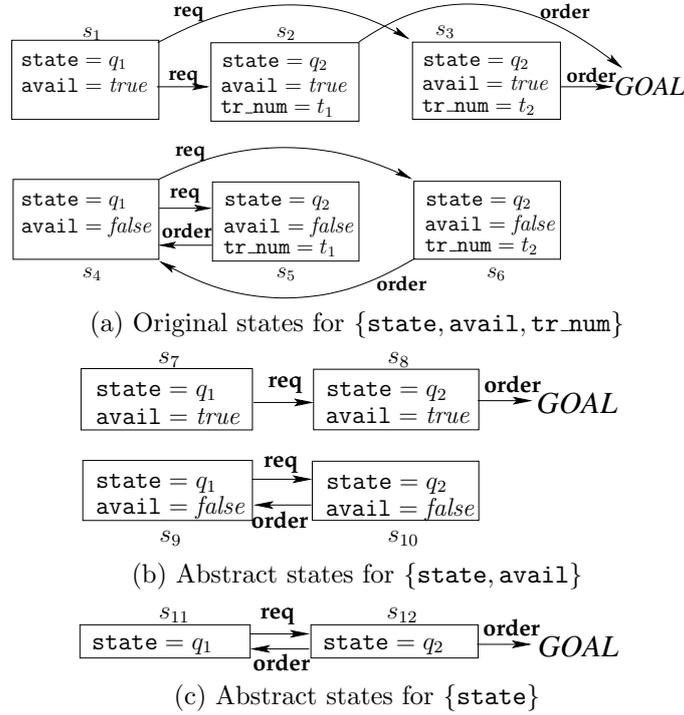


Figure 3 Abstraction

## 7.2 Abstraction and refinement algorithm

Algorithm 2 presents a high-level description of our method based on signature-preserving abstraction. In a nutshell, we abstract a given web services  $W$  into  $W'$  and try to find a solution for the abstraction  $W'$ . If we identify such a coordinator, it can indeed control the original web services  $W$  to satisfy a given goal. Otherwise, we repeat the search with more accurate abstraction.

First, we abstract  $W$  with only input and output variables, i.e.,  $Y = X^I \cup X^O$  (lines 1–2). We then try to construct a solution coordinator for the abstraction  $W_Y$  (line 3). If we find a coordinator  $c$  such that  $W_Y || c \models G$ , then  $c$  also satisfies  $W || c \models G$  by Theorem 4. Otherwise, we refine our current abstraction  $W_Y$  by adding more variables, and try to find  $c$  for the new abstraction (lines 7–13). How to select additional variables will be elaborated in Section 7.3. We repeat the abstraction/refinement step until we identify a coordinator  $c$  satisfying  $W_Y || c \models G$  or the variable set used for abstraction equals to the original variable set. The latter case implies that no solution exists for the given problem. For the procedure  $WSC_{PartialObs}$  we use in this algorithm, we can employ the algorithm for WSC with partial observation in Algorithm 1, of which the complexity is  $O(2^{2^n})$ . Once we identify a solution by using abstract web services which have less variables than the given web services, searching space required is shrunken (double-)exponentially in the number of variables that we save.

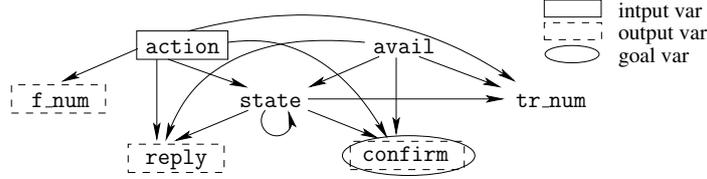
**Algorithm 2:** Signature-preserving Abs/Ref WSC

---

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

- 1  $Y := X^I \cup X^O$ ;
- 2  $W_Y := \text{Abstraction}(W, Y)$
- 3 **if**  $((c := \text{WSCPartialObs}(W_Y, G)) \neq \text{null})$  **then**
- 4 | **return**  $c$ ;
- 5 **end if**
- 6  $\text{ConstructDependencyGraph}(W, G)$ ;
- 7 **while**  $((\text{newVars} := \text{SelectNewVars}(W, G)) \neq \text{null})$  **do**
- 8 |  $Y := Y \cup \text{newVars}$ ;
- 9 |  $W_Y := \text{Abstraction}(W, Y)$ ;
- 10 | **if**  $((c := \text{WSCPartialObs}(W_Y, G)) \neq \text{null})$  **then**
- 11 | | **return**  $c$ ;
- 12 | **end if**
- 13 **end while**
- 14 **return**  $\text{null}$ ;

---

**Figure 4** Variable dependency graph

### 7.3 Automatic refinement

Failure to identify a coordinator for abstract web services (line 3 or 9 in Algorithm 2) is caused either by too coarse abstraction or by the fact that a coordinator for the original web services does not exist. For the latter case, since we check it with the original web services in the worst case, Algorithm 2 will correctly conclude that there is no solution.

**Theorem 5 (Completeness)** *Given a set of web services  $W$  and a goal  $G$ , if there does not exist a coordinator  $c$  to satisfy  $W \parallel c \models G$ , Algorithm 2 eventually returns  $\text{null}$ .*

However, in the former case, although there exists a coordinator for the original web services  $W$ ,  $\text{WSCFullObs}$  or  $\text{WSCPartialObs}$  returns  $\text{null}$  for the abstraction  $W_Y$ . The reason is that removing too many variables, including ones with significant information to reach a goal, gives too much freedom to the abstraction. It induces some infeasible paths to states not satisfying the goal. For instance, in Figure 3(c), since we remove the variable **avail**,  $s_1$  ( $s_2$  and  $s_3$ ) is indistinguishable from  $s_4$  ( $s_5$  and  $s_6$ , respectively). Thus, an infeasible edge from  $s_{12}$  to  $s_{11}$  by **order** is introduced, by which we no longer have a strategy to guarantee to reach a goal.

Therefore, we have to refine the current abstraction to find a solution by adding more variables. Since the infeasible paths to states not satisfying the goal prevent us from identifying a solution coordinator, it is important to accurately keep track of the values of variables appearing in the given goal predicate. Given this, the most significant criterion for selecting variables to be added is the relevance to variables in the goal predicate. To evaluate each variable's relevance to the goal variables, we construct a variable dependency graph.

**Definition 6 (Variable dependency graph)** *Given a set of web services  $W$  and a goal  $G$ , a variable dependency graph is a directed graph  $G(V, E)$  where a set  $V$  of vertices is  $\{x \mid x \in X \cup X^I\}$  and a set  $E$  of directed edges is  $\{(x \triangleright y) \mid x, y \in V, \text{the value of } y \text{ depends on the value of } x\}$ .*

For instance, the pseudo-codes “ $y := x$ ” and “**if** ( $x = \text{true}$ ) **then**  $y := 0$ ” imply that the value of  $y$  depends on  $x$ . Figure 4 illustrates a fraction of the variable dependency graph for  $W$  and  $G$  in Example 4. It shows only variables of  $w_{AR}$ . For example, since the values of `state`, `reply` and `tr_num` depend on the values of `state`, `action` and `avail` (see the first part of  $T$  in Example 2), we have corresponding directed edges  $(\text{state} \triangleright \text{state})$ ,  $(\text{action} \triangleright \text{state})$ ,  $(\text{avail} \triangleright \text{state})$ ,  $\dots$ , and  $(\text{action} \triangleright \text{tr\_num})$  in Figure 4. In the dependency graph, it is clear that variables with stronger dependency to the variables in the goal predicate have a shorter hop to the goal variables. Thus, in each iteration of Algorithm 2, the procedure *SelectNewVars* returns a set of variables that have the closest hop to the variables in the goal predicate (i.e., 1-hop, 2-hop, and so on). For instance, since `confirm` is a variable in the goal predicate, the set of variables that have 1-hop dependency is  $\{\text{action}, \text{avail}, \text{state}\}$ .

## 8 Signature-subsuming Abstraction

In Section 7, we restricted the target of abstraction to internal variables; namely, abstract web services have the same I/O variables as the original ones. However, in many cases, we have observed that some of output variables do not provide any important information for a coordinator to decide its move. For instance, the airline reservation web service in Example 2 simply copies the request value (i.e.,  $req_1$  and  $req_2$ ) to the flight number (i.e.,  $f_1$  and  $f_2$ ), and returns it to clients for reference. In this case, even without this output, the coordinator can successfully control the given web services to satisfy the goal. In this section, we consider, as the target of abstraction, output variables as well as internal variables.

First, we define *signature-subsuming abstract web services* for given web services, which have the same input variables, but less internal variables and output variables.

**Definition 7 (Signature-subsuming abstract web services)** *Given a set of web services  $W(X, X^I, X^O, \text{Init}, T)$ , and a set  $Y$  of variables such that  $X^I \subseteq Y \subseteq X^A$ , the signature-subsuming abstraction of  $W$  with respect to  $Y$  is  $W_Y(X_Y, X_Y^I, X_Y^O, \text{Init}_Y, T_Y)$  where  $X_Y = Y \setminus X^I$ ,  $X_Y^I = X^I$ ,  $X_Y^O = Y \cap X^O$ , and  $\text{Init}_Y$  and  $T_Y$  are defined as the same as Definition 5.*

**Table 1** Experiment result

Problem	T var	I/O var	Basic	Sig-pres	Sv var	Sig-sub	Sv var
TAS-a	38	9	5.8	2.9	6	<b>0.1</b>	6/4
TAS-b	42	8	61.4	55.3	2	<b>13.8</b>	2/1
TAS-c	69	10	>7200.0	>7200.0	6	<b>162.0</b>	6/2
P&S-a	44	9	50.4	49.8	11	<b>3.2</b>	11/2
P&S-b	55	10	320.0	364.6	19	<b>42.3</b>	19/3
P&S-c	63	10	>7200.0	>7200.0	20	<b>1214.0</b>	20/3
VOS-a	61	15	208.3	195.7	14	<b>18.2</b>	14/4
VOS-b	74	15	3323.0	2321.3	23	<b>520.8</b>	23/4

Since signature-subsuming abstract web services  $W_Y$  have less output variables than the original web services  $W$ , any coordinator  $c$  which can be composed with  $W_Y$  is also able to be composed with  $W$  by ignoring redundant output variables of  $W$  (i.e., ignoring  $X^O \setminus X_Y^O$ ). Moreover, since  $W_Y$  contains all the behaviors of  $W$ , Theorem 4 is still valid.

For selecting output variables to be used in abstraction, we again employ the variable dependency graph in Section 7.3. In general, output variables which depend on internal variables that in turn depend on variables in a goal predicate tend to provide important information on the state of web services for the coordinator to control the web services. For instance, in Figure 4, `reply` has a dependency on `state` and `avail` that have a dependency on the goal variable `confirm`, and `reply` is an important output by which a coordinator infers whether a flight seat is available. On the other hand, `f_num` that represents a flight number has dependency only on an input variable, `action`, and it does not provide any information to help a coordinator. Therefore, we find such a set  $X^{SO} \subseteq X^O$  of *significant output variables* which have a dependency on internal variables with a dependency on variables in a goal predicate, and then use  $X^{SO}$  for the initial abstraction. That is, in signature-subsuming abstraction, we start  $Y := X^I \cup X^{SO}$  as line 1 in Algorithm 2. The rest of output variables (i.e.,  $X^O \setminus X^{SO}$ ) are used in the last iteration.

## 9 Empirical Validation

We have implemented automatic tools for signature-preserving/signature-subsuming abstraction and refinement, using a state-of-the-art planning tool, MBP [6]. Given a set of web service descriptions in WS-BPEL files, and a goal predicate, our tools automatically construct a coordinator web service which can control the given web services to achieve the goal. To demonstrate that our tools efficiently synthesize coordinators, we compared the basic algorithm [43] and our methods with 3 sets of realistic examples (8 instances); Travel agency system (TAS), Producer and shipper (P&S), and Virtual online shop (VOS). Since there are no public benchmark test sets, we have selected web service examples popularly used in web service composition research. TAS was explained in Example 1. We have three instances, TAS-a, TAS-b, and TAS-c, where we have 4, 9, and 16 options, respectively, for input values for flight reservation and hotel reservation each. Producer and shipper (P&S) [43, 32] includes two web services, Producer and Shipper. Producer produces furniture items, and Shipper delivers an item from

an origin to a destination. We have three instances, P&S-a, P&S-b, and P&S-c where there are 4, 6, and 8 options, respectively, for furniture order and delivery order each. The virtual online shop (VOS) example [4] includes Store and Bank web services where Store sells items and Bank transfers money from one account to another account. This example includes two instances, VOS-a and VOS-b where there are 3 and 4 options, respectively, for item orders and money transfer each.

All experiments were performed on a PC using a 2.4GHz Pentium processor, 2GB memory and a Linux operating system. Table 1 presents the number of total boolean variables (T var) and input/output variables (I/O var). It also shows the total execution time in seconds for the basic algorithm (Basic) and our methods, i.e., Signature-preserving (Sig-pres) and Signature-subsuming (Sig-subs), and the number of boolean variables that we saved (Sv var). In the signature-subsuming case, the table presents the number of internal variables/IO variables which we saved. Our experiment shows that our technique outperforms the basic algorithm in terms of execution time. The numbers of iterations in our experiments were around 2–3, since variable dependency graphs were relatively shallow. In WSC literature, in general, behavior descriptions in WS-BPEL or OWL-S tend *not* to be complex, which usually yields to shallow variable dependency graphs.

Although we have employed modest size of examples, our abstraction technique can be useful even for larger size examples since in general, the number of variables which have relevance with goal variables is limited. In such a case, our techniques can eliminate a number of irrelevant variables, improving the convergence speed considerably.

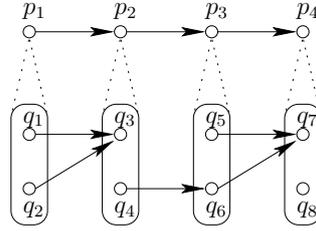
## 10 Discussion

In this section, we discuss our refinement process presented in Section 7.3, comparing it with the *counter-example guided abstraction refinement* [11] in model checking literature.

### 10.1 Refinement in model checking

Model checking is an automatic technique to verify that a system satisfies its specification, where the system is represented as a Kripke structure and the specification is described in a temporal logic, e.g., linear temporal logic (LTL) [33] or computational tree logic (CTL) [13]. Model checking has been successfully applied in hardware verification, and is emerging as an industrial standard tool for H/W design. However, one of the main challenges in model checking is the *state explosion problem* which can occur when a system includes many parallel components. A number of approaches have been studied to reduce the number of states in the model. Among them, *abstraction* [12, 11] is considered as the most general and flexible method for handling the state explosion problem.

Clarke et al. [11] proposed a fully automatic technique called *counter-example guided abstraction refinement*. This method starts with a relatively small abstraction of a system to be verified. If this abstraction satisfies a given safety property, we can conclude that the original system also satisfies the property since we use *over-approximation* in this technique. Otherwise, the technique provides a



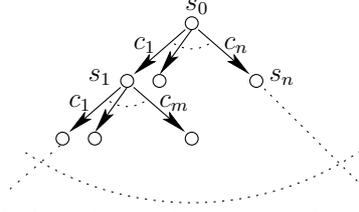
**Figure 5** Spurious counter-example in model checking

counter-example showing that the abstract model violates the given property. In this case, however, with this counter-example we cannot conclude whether the original system violates the property or not. Therefore, we check if the counter-example is feasible also in the original system. If so, we can declare that the original system violates the given property. Otherwise, the counter-example path is spurious in the original system, and this path is exploited for refinement step. Figure 5 illustrates a spurious counter-example. Let us assume that we are given a counter-example path  $p_1p_2p_3p_4$  for an abstract system, and each state  $p_1, p_2, p_3$ , and  $p_4$  corresponds to a set of states of the original system,  $\{q_1, q_2\}$ ,  $\{q_3, q_4\}$ ,  $\{q_5, q_6\}$ , and  $\{q_7, q_8\}$ , respectively. By traversing the original system, it is easy to see if the counter-example path is spurious. The reason why this infeasible path is introduced is that  $q_3$  and  $q_4$  are mapped into the same abstract state  $p_2$ . That is, the concrete state  $q_3$  is reachable from the initial state, but it does not have any outgoing transition to the next state in the counter-example. On the other hand,  $q_4$  is not reachable but it has an outgoing transition which causes the spurious counter-example. Therefore, in the refinement step in model checking, we automatically identify a new abstraction so that these two states do not correspond to the same abstract state. Finally, we repeat the model checking step with the new abstraction.

### 10.2 Refinement in behavioral description based WSC

The behavioral description based WSC presented in this paper has a similar high-level structure. We start with a small abstraction of a given set of web services. If we can identify a coordinator web service that controls the *abstract* web services to reach a given goal state, our algorithm returns the coordinator web service. This coordinator is able to control the *original* web services since we use *over-approximation* (Theorem 4). Otherwise, our tool provides a counter-example which demonstrates that there is no coordinator to control the abstract web services. In this case, as counter-example guided abstraction refinement in model checking, we have to check whether there is indeed no coordinator to control the original web services *or* the counter-example is spurious.

However, the main difference between “counter-example guided abstraction refinement in model checking” and “behavioral description based WSC” is the counter-example structure. Since the purpose of the former is to check a safety property of the system, a counter-example is a *path* from the initial state to a state which violates the safety property (see Figure 5). On the other hand, since the purpose of the latter is to construct a coordinator which can control a given set of web services, a counter-example is a *tree* to ensure that for every



**Figure 6** Counter-example for behavioral description based WSC

choice of the coordinator, we cannot enforce the given web services to eventually reach a goal state. Figure 6, for instance, illustrates a counter-example tree for behavioral description based WSC problem. Each node in the counter-example tree corresponds to a state of a given web services. For each node, we have a set of choices of the coordinator, e.g.,  $c_1 \cdots c_n$  at the state  $s_0$  in Figure 6. The counter-example tree demonstrates that for any choice there is no path to eventually reach a goal state.

Now, in order to check whether there is indeed no coordinator to control the original web services *or* the counter-example tree is spurious due to the over-approximation, we have to traverse the abstract web services with the given counter-example tree. The main obstacle for this test is that, in general, the counter-example tree is much larger than the counter-example path in model checking. That is, the number of nodes in the tree is, in the worst case,  $2^{2^n}$  where  $n$  is the number of total boolean variables in a given set of web services since a coordinator has to control them with *partial observation*. Hence, in our technique, we avoid the feasibility checking and add variables with strong dependency on goal variables to the next abstraction. We then try to identify a coordinator for the new abstraction.

## 11 Conclusion

In this paper, we proposed approximation based techniques to automatically construct a coordinate web service that controls a given set of web services to reach a goal state, which is a computationally hard problem (i.e., 2-EXP-hard in general). The proposed solutions first reduce the given web services into abstract web services with less variables using over-approximation. We then try to find a coordinate web service for the abstract web services. If this trial succeeds, the found coordinator can also control the original web services since we apply over-approximation. Otherwise, we add important variables for the next abstraction. Our experimental results show promising results, validating the effectiveness of our proposals.

Several directions lie ahead for future work. First, we plan to study other abstraction methods and refinement techniques for the early convergence of the conclusion. Second, we will extend our technique for the WSC problem with more expressive goals (e.g., goals specified in temporal logics). Third, we want to study a tight bound for variables required to solve this problem. Finally, it is also an interesting issue to integrate ontological knowledge to constrain the behavioral description-based WSC problem.

## Acknowledgements

This research was partially supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency): NIPA-2012-H0301-12-3006. Dongwon Lee was partially supported by NSF DUE-0817376 and DUE-0937891 awards.

## References

- [1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture, and Applications*. Springer-Verlag (ISBN: 3540440089), 2003.
- [2] M. Antoniotti. *Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the control-D system*. PhD thesis, New York University, 1995.
- [3] G. Armano, G. Cherchi, and E. Vargiu. A parametric hierarchical planner for experimenting abstraction techniques. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 936–941, 2003.
- [4] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *IEEE International Conference on Web Services (ICWS2006)*, pages 63–71, 2006.
- [5] D. Berardi, D. Calvanese, D. Giacomo, Giuseppe, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *Int. J. of Cooperative Information Systems*, 14(4):333–376, 2005.
- [6] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artificial Intelligence*, 170(4-5):337–384, 2006.
- [7] P. Bertoli and M. Pistore. Planning with extended goals and partial observability. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 270–278, 2004.
- [8] Bindingpoint. <http://www.bindingpoint.com> (last accessed may 1, 2006).
- [9] A. Brogi. On the potential advantages of exploiting behavioural information for contract-based service discovery and composition. *Journal of Logic and Algebraic Programming*, pages 1–10, 2010.
- [10] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
- [11] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

- [12] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.
- [13] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990.
- [14] eSynaps. <http://www.robtext.com/dns/esynaps.com.html>.
- [15] W. Fan, F. Geerts, W. Gelade, F. Neven, and A. Poggi. Complexity and composition of synthesized web services. In *PODS '08: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 231–240, 2008.
- [16] H. Haas. *Architecture and future of web services: from SOAP to semantic web services*. 2004.
- [17] A. Herzig, J. Lang, D. Longin, and T. Polacsek. A logic for planning under partial observability. In *Proceedings of National Conference on Artificial Intelligence (AAAI-00)*, pages 768–773, 2000.
- [18] W. Huang, Z. Wen, Y. Jiang, and L. Wu. Observation reduction for strong plans. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1930–1935, 2007.
- [19] R. Hull. E-service composition: Models and formalisms. In *Description Logics*, 2003.
- [20] H. Kil, W. Nam, and D. Lee. Computational complexity of web service composition based on behavioral descriptions. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, pages 359–363, 2008.
- [21] H. Kil, W. Nam, and D. Lee. Efficient abstraction and refinement for behavioral description based web service composition. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1740–1745, 2009.
- [22] O. Kupferman and M. Vardi. Synthesis with incomplete information. In *Proceedings of the 2nd International Conference on Temporal Logic*, pages 91–106, 1997.
- [23] S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *KR*, pages 482–496, 2002.
- [24] H. Meyer, D. Kuropka, and P. Tröger. ASG—techniques of adaptivity. In *Autonomous and Adaptive Web Services*, 2007.
- [25] M. Moffitt. On the partial observability of temporal uncertainty. In *Proceedings of National Conference on Artificial Intelligence (AAAI-07)*, pages 1031–1037, 2007.

- [26] W. Nam, H. Kil, and 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)*, pages 331–334, 2008.
- [27] W. Nam, H. Kil, and D. Lee. On the computational complexity of behavioral description-based web service composition. *Theoretical Computer Science*, 412(48):6736–6749, 2011.
- [28] Uddi version 3.0.2. <http://uddi.xml.org/>, 2004.
- [29] Web services business process execution language version 2.0. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>, 2007.
- [30] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- [31] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1252–1259, 2005.
- [32] M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pages 2–11, 2005.
- [33] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
- [34] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1989)*, pages 179–190, 1989.
- [35] J. Reif. The complexity of two-player games of incomplete information. *Journal on Computer and System Sciences*, 29:274–301, 1984.
- [36] RemoteMethods. <http://www.remotemethods.com>.
- [37] J. Rintanen. Complexity of planning with partial observability. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 345–354, 2004.
- [38] M. T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A scalable and ontology-based p2p infrastructure for semantic web services. In *Peer-to-Peer Computing*, pages 104–111, 2002.
- [39] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.
- [40] T. Smith, D. R. Thompson, and D. Wettergreen. Generating exponentially smaller pomdp models using conditionally irrelevant variable abstraction. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS2007)*, pages 304–311, 2007.

- [41] Speed-r: Semantic p2p environment for diverse web service registries. <http://webster.cs.uga.edu/mulye/SemEnt/Speed-R.html>.
- [42] W. Thomas. Infinite games and verification. In *Proceedings of the International Conference on Computer Aided Verification (CAV'02)*, LNCS 2404, pages 58–64. Springer, 2002.
- [43] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *Proceedings of International Semantic Web Conference (ISWC'04)*, pages 380–394, 2004.
- [44] M. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *Proceedings of International Conference on Computer Aided Verification (CAV'95)*, pages 267–278, 1995.
- [45] xMethod. <http://www.xmethod.com>.
- [46] J. Yu, Y. Han, J. Han, Y. Jin, P. Falcarin, and M. Morisio. Synthesizing service composition models on the basis of temporal business rules. *Journal of Computer Science and Technology*, 23(6):885–894, 2008.