Efficient Abstraction and Refinement for Behavioral Description Based Web Service Composition

Hyunyoung Kil **Wonhong Nam** **Dongwon Lee**
The Pennsylvania State University
{hykil, wnam, 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, c, 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 c has only *partial observations*) remains to be *doubly exponential* in the number of variables in web service descriptions, rendering any 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*. We empirically validate that our proposals can solve realistic problems efficiently.

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 is, given a set of (behavioral descriptions of) web services, W, and a reachability goal, G, 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 what the web service executes internally and externally with 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 researches on the WSC problem, only a few (e.g., [Traverso and Pistore, 2004; Pistore *et al.*, 2005a; 2005b]) employ realistic models with partial observation. Our previous work [Kil *et al.*, 2008] investigated the computational complexity (i.e., lower bound) of the WSC problem: (1) solving the WSC problem for a restricted case (when the synthesized coordinator web service, c, has *full observation* for all variables of the given web services) is EXP-hard, and (2) solving the WSC problem for a general case (when c has *partial observation*) is 2-EXP-hard. These results suggest studying efficient approximation solutions to the WSC problem. Toward this challenge, in this paper, we propose two approximation-based algorithms using "abstraction and refinement" [Clarke *et al.*, 1994]. 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* which has a strong connection with WSC, 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 abstraction, we propose two methods—*signature-preserving abstraction* and *signature-subsuming abstraction*. We report on the performance of our tool on 3 sets of realistic problems (8 instances), comparing with a basic algorithm [Traverso and Pistore, 2004] without abstraction/refinement. Our experiment shows that our technique 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 [Pistore *et al.*, 2005b].

2 Web Service Composition & Lower Bounds

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 up both a flight ticket and a hotel room. Figure 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. If there is an available accommodation, it returns the room information and its price; otherwise, it returns "Not Available". AR then pro-

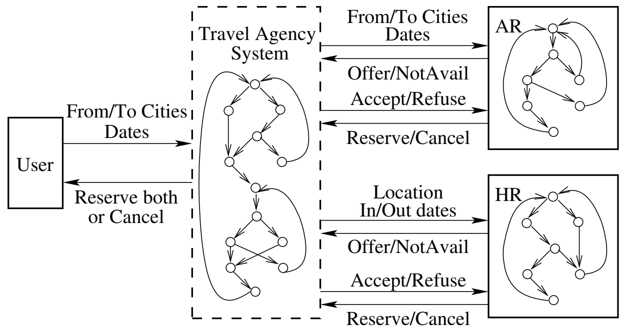


Figure 1: Travel agency system

cesses a reply "Accept" or "Refuse" from its environment. □

Definition 1 (Web service) A (behavioral description of) *web service* w is a 5-tuple $(X, X^I, X^O, \mathit{Init}, T)$ where:

- X is a finite set of *variables* that w controls. A state s of w is a valuation for every variable in X. We denote a set of all states 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 $x \in X \cup X^I$ 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 its environment 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., $X^A = X \cup X^I$).
- $\mathit{Init}(X)$ is an *initial predicate* over X. $\mathit{Init}(s) = \mathit{true}$ 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 $X' = \{x' \mid x \in X\}$, 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. [Traverso and Pistore, 2004] is based on an explicit state-transition system using a set of states, we define *symbolically* 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 $(X, X^I, X^O, \mathit{Init}, T)$ where:

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

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. 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 $W = \{w_1, \cdots, w_n\}$ of web services where each w_i is $(X_i, X_i^I, X_i^O, \mathit{Init}_i, T_i)$, and X_i and X_i^I are disjoint with each other X_j and X_j^I, respectively, W also can be represented by a 5-tuple $(X, X^I, X^O, \mathit{Init}, T)$ where $X = X_1 \cup \cdots \cup X_n$, $X^I = X_1^I \cup \cdots \cup X_n^I$, $X^O = X_1^O \cup \cdots \cup X_n^O$, $\mathit{Init}(X) = \mathit{Init}_1 \wedge \cdots \wedge \mathit{Init}_n$, and $T(X, X^I, X') = T_1 \wedge \cdots \wedge T_n$. □

Since a *coordinator web service* is also a web service, it is a 5-tuple $c(X_c, X_c^I, X_c^O, \mathit{Init}_c, T_c)$. 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 $T_c(s_c, in, s_c') = \mathit{true}$.

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 $W(X, X^I, X^O, \mathit{Init}, T)$ of web services and a coordinator $c(X_c, X_c^I, X_c^O, \mathit{Init}_c, T_c)$ where $X^I = X_c^O$ 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 in $W||c$ is in $S \times S_c$.
- The root node is (s, s_c) such that $\mathit{Init}(s) = \mathit{true}$ and $\mathit{Init}_c(s_c) = \mathit{true}$.
- For each node (s, s_c), it has a set of child nodes, $\{(s', s_c') \mid T(s, in, s') {=} \mathit{true}, in {=} s_c[X^I], T_c(s_c, in_c, s_c') {=} \mathit{true}, in_c {=} s'[X^O]\}$. □

In the above, intuitively, the web services W, by receiving the input in from the current state s_c of the coordinator, collectively proceeds 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 τ-transition [Pistore *et al.*, 2005a]. A *goal* $G \subseteq S$ is a set of states to reach, and specified as a predicate. Given a set W of web services, a coordinator c, and a goal G, we define $W||c \models G$ if for every path $(s^0, s_c^0)(s^1, s_c^1) \cdots$ 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.

Definition 4 (Web service composition problem) 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 $W||c \models G$. $\square$

Example 3. In Example 1, we wish to reserve both a flight ticket and a hotel room. This can be represented as $G \equiv (\texttt{flightConfirm} = \mathit{reserve}) \wedge (\texttt{hotelConfirm} = \mathit{reserve})$. 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 $W||c \models G$. $\square$

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 $W(X, X^I, X^O, \mathit{Init}, T)$ 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.

Theorem 1. *The WSC problem with full observation is exponential in the number of variables in W.* $\blacksquare$

The proof is to simulate an *alternating Turing machine (ATM)* [Papadimitriou, 1994] with a *polynomial* space bound. That is, for any ATM A and any input string σ, we can construct a WSC problem in polynomial time such that A accepts σ if and only if there exists a coordinator to satisfy a goal.

Theorem 2. *The WSC problem with partial observation is doubly-exponential in the number of variables in W.* $\blacksquare$

The proof is to simulate an ATM with *exponential* space bound. For the details of both proofs, see [Kil *et al.*, 2008].

3 Basic Algorithm for WSC Problem

In this section, we study a basic algorithm for the general WSC problem defined in Section 2. Several researches [Traverso and Pistore, 2004; Pistore *et al.*, 2005a] have successfully applied a planning technique with partial observation [Bertoli *et al.*, 2006] 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 [Bertoli *et al.*, 2006]. In a general case of WSC, a coordinator web service is not able to identify the exact state of target web services. Hence, we model this uncertainty by using a *belief state*, which is a set of *possible* states of target web services but *indistinguishable*. The underlying idea of Algorithm 1 is to construct an *and-or searching tree* 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 coordinator's 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. Next (lines 5–12), until

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  while ($tree.root.result = undecided$) do
6      $node := SelectNode(tree)$;
7      $childNodes := ExtendTree(tree, node)$;
8      if ($CheckSuccess(childNodes)$) then
9          $node.result := true$;
10     else if ($CheckFailure(childNodes)$) then
11         $node.result := false$;
12     $PropagateResult(tree, node)$;
13 if ($tree.root.result = true$) then
14     return $ConstructCoordinator(tree)$;
15 else return $null$;
```

determining the result value for the root, we repeat: (1) to select a node which is not determined yet as "*true*" or "*false*", (2) to extend the tree from the selected node by computing a set of possible successor nodes, and (3) to check if the node can reach a goal state based on the and-or constraint. Once we identify the result of each node, we propagate the result to its ancestor node. 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. Otherwise, it returns *null*. 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 2).

4 Signature-preserving Abstraction and Refinement

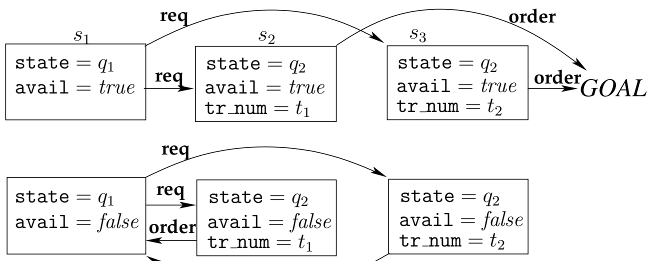
Theorems 1 and 2 imply that the WSC problem is computationally hard. Hence, more efforts to devise efficient approximation solutions to the WSC problem are needed. In addition, the complexity of Algorithm 1 also provides the same implication. Therefore, we propose two approximation-based methods using abstraction and refinement in Sections 4 and 5.

4.1 Signature-preserving abstraction

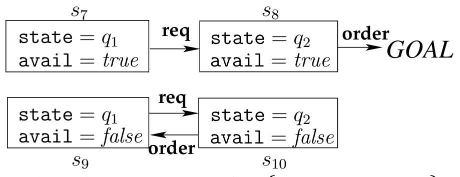
Given a set W of web services, we define *signature-preserving abstract web services* that have the same signature (i.e., the same I/O variables) but less variables than W.

Definition 5 (Signature-preserving abstract web services) Given a set of web services $W(X, X^I, X^O, \mathit{Init}, T)$ 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 $W_Y(X_Y, X^I_Y, X^O_Y, \mathit{Init}_Y, T_Y)$ where:

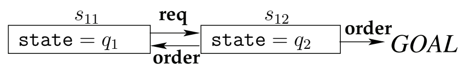
- $X_Y = Y \setminus X^I$, $X^I_Y = X^I$, and $X^O_Y = X^O$.
- For every $s_Y \in S_Y$, $\mathit{Init}_Y(s_Y) = \mathit{true}$ iff $\exists s \in S.\ (\mathit{Init}(s) = \mathit{true}) \wedge (s_Y = s[X_Y])$.
- For every $s_Y, s'_Y \in S_Y$, $T_Y(s_Y, in, s'_Y) = \mathit{true}$ iff $\exists s, s' \in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



(a) Original states for {state, avail, tr_num}



(b) Abstract states for {state, avail}



(c) Abstract states for {state}

Figure 2: Abstraction

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 3 (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 4 (Abstraction). Figure 2(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 2(b) shows an abstract state space with respect to {state, 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 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 2(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. □

4.2 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.

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  $W_Y := Abstraction(W, Y)$;     // $W_Y$ has only $X^I$ and $X^O$.
3  if (($c := WSCFullObs(W_Y, G)) \neq null$) then
4      return $c$;
5  $ConstructDependencyGraph(W, G)$;
6  while (($newVars := SelectNewVars(W, G)) \neq null$) do
7      $Y := Y \cup newVars$;
8      $W_Y := Abstraction(W, Y)$;
9      if (($c := WSCPartialObs(W_Y, G)) \neq null$) then
10         return $c$;
11 return $null$;
```

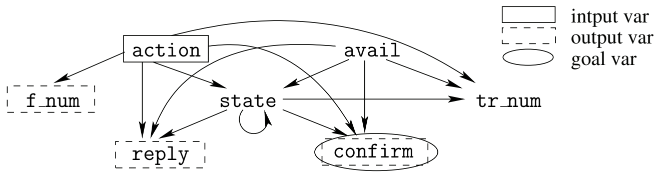


Figure 3: Variable dependency graph

First, we abstract W with only input and output variables, i.e., $Y = X^I \cup X^O$ (lines 1–2). Since, at this point, W_Y does not include any internal variable (i.e., $X_Y = X_Y^O$), we can exploit, in this case, the algorithm for WSC with full observation, *WSCFullObs*, which is more efficient algorithm (EXP-hard). For the sake of space, we do not show the details of *WSCFullObs*. If we find a coordinator c such that $W_Y||c \models G$, then c also satisfies $W||c \models G$ by Theorem 3. Otherwise, we refine our current abstraction W_Y by adding more variables, and try to find c for the new abstraction (lines 6–10). How to select additional variables will be elaborated in Section 4.3. We repeat the abstration/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. Although from the second loop, we should employ the algorithm for WSC with partial observation, *WSCPartialObs*, with $O(2^{2^n})$ complexity, once we identify a coordinator using small abstract web services, searching space is shrunken (double-)exponentially in the number of variables that we save.

4.3 Automatic refinement

If we fail to identify a coordinator for abstract web services (line 3 or 9 in Algorithm 2), it 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 4 (Completeness). *Given a set of web services W and a goal G, if there does not exist a coordinator c to satisfy $W||c \models G$, Algorithm 2 eventually returns $null$.* ■

However, in the former case, although there exists a coordinator for the original web services W, *WSCFullObs* or *WSCPartialObs* returns *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 2(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. With this reason, 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 vertexes is $\{x \mid x \in X \cup X^I\}$ and a set E of *directed* edges is $\{(x \triangleright y) \mid x, y \in V$, the value of y depends on the value of $x\}$. $\square$

For instance, the pseudo-codes "y := x" and "**if** (x = *true*) **then** y :=0" imply that the value of y depends on x. Figure 3 illustrates a fraction of the variable dependency graph for W and G in Example 3. 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 (`state`$\triangleright$`state`), (`action`$\triangleright$`state`), (`avail`$\triangleright$ `state`), $\cdots$, and (`action` $\triangleright$ `tr_num`) in Figure 3. In the dependency graph, it is clear that variables with stronger dependency to the variables in the goal predicate locate closer 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 $\{$`action`, `avail`, `state`$\}$.

5 Signature-subsuming Abstraction

In Section 4, we restricted the target of abstraction to internal variables; namely, abstract web services have the same I/O variables with 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., *req₁* and *req₂*) to the flight number (i.e., *f₁* and *f₂*), and returns it to clients for reference. In this case, even without this output, the coordinator can successfully control given web services to satisfy the goal. Hence, 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 $W(X, X^I, X^O, \mathit{Init}, T)$, 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 $W_Y(X_Y, X^I_Y, X^O_Y, \mathit{Init}_Y, T_Y)$ where $X_Y = Y \setminus X^I$, $X^I_Y = X^I$, $X^O_Y = Y \cap X^O$, and Init_Y and T_Y are defined as the same as Definition 5. $\square$

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 $X^O \setminus X^O_Y$). Moreover, since W_Y contains all the behaviors of W, Theorem 3 is still valid.

For selecting output variables to be used in abstraction, we again employ the variable dependency graph in Section 4.3. In general, output variables that 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 3, `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 infer 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.

6 Empirical Validation

We have implemented automatic tools for signature-preserving/signature-subsuming abstraction and refinement, using a state-of-the-art planning tool, MBP [Bertoli *et al.*, 2006]. 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 [Traverso and Pistore, 2004] 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 researches. 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) [Traverso and Pistore, 2004; Pistore *et al.*, 2005a] 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 de-

Table 1: Experiment result

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

livery order each. Virtual online shop (VOS) [Barbon *et al.*, 2006] 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 have been performed on a PC using a 2.4GHz Pentium processor, 2GB memory and a Linux operating system. Table 1 presents the number of total variables (Total var) and input/output variables (I/O var) in boolean. It also shows the total execution time in seconds for the basic algorithm (Basic) and our methods (Signature-preserving and Signature-subsuming), and the number of boolean variables that we saved (Saved 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. For instance, with 100 web services with 10 variables each (total 1,000 variables), a goal that users want is often associated with only a fraction of available web services and variables (say 5%, 50 variables). In such a case, our techniques can eliminate 95% of irrelevant variables, improving the convergence speed considerably.

7 Related Work and Conclusion

In web service compositions, many researches [Traverso and Pistore, 2004; Pistore *et al.*, 2005a; 2005b; Nam *et al.*, 2008] have been carried out, but only a few ones employ realistic models with partial observability. [Traverso and Pistore, 2004; Pistore *et al.*, 2005a; 2005b] have defined web service compositions with partial observability, and presented algorithms and tools using their automated planning techniques. However, to the best of our knowledge, there is no study for WSC problems or planning on partial observation using abstraction and refinement.

The WSC problem has a strong connection with automated planning under partial observation. In [Bertoli *et al.*, 2006], a fully automatic planning tool MBP has been developed for this setting based on belief-states. On the other hand, several researches have been performed in planning using abstraction. Huang et al. [Huang *et al.*, 2007] propose an algorithm to reduce observation variables for strong plans. This technique, however, cannot identify such a variable until a plan is constructed. Thus, it cannot be applied to our problem. Armano et al. [Armano *et al.*, 2003] employ abstraction techniques for a hierarchical planner. Smith et al. [Smith *et al.*, 2007] present an abstraction technique to generate exponentially smaller POMDP.

In this paper, we proposed approximation based techniques for WSC problems based on abstraction and refinement. Our preliminary experiment showed promising results. Several directions are ahead for future work. First, we plan to study other abstraction methods and refinement techniques to early converge 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.

References

[Armano *et al.*, 2003] G. Armano, G. Cherchi, and E. Vargiu. A parametric hierarchical planner for experimenting abstraction techniques. In *IJCAI*, pages 936–941, 2003.

[Barbon *et al.*, 2006] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *ICWS*, pages 63–71, 2006.

[Bertoli *et al.*, 2006] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artificial Intelligence*, 170(4):337–384, 2006.

[Clarke *et al.*, 1994] E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[Huang *et al.*, 2007] W. Huang, Z. Wen, Y. Jiang, and L. Wu. Observation reduction for strong plans. In *IJCAI*, pages 1930–1935, 2007.

[Kil *et al.*, 2008] H. Kil, W. Nam, and D. Lee. Computational complexity of web service composition based on behavioral descriptions. In *ICTAI*, pages 359–363, 2008.

[Nam *et al.*, 2008] W. Nam, H. Kil, and D. Lee. Type-aware web service composition using boolean satisfiability solver. In *CEC/EEE*, pages 331–334, 2008.

[Papadimitriou, 1994] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pistore *et al.*, 2005a] M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *ICAPS*, pages 2–11, 2005.

[Pistore *et al.*, 2005b] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *IJCAI*, pages 1252–1259, 2005.

[Smith *et al.*, 2007] T. Smith, D. R. Thompson, and D. Wettergreen. Generating exponentially smaller POMDP models using conditionally irrelevant variable abstraction. In *ICAPS*, pages 304–311, 2007.

[Traverso and Pistore, 2004] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *ISWC*, pages 380–394, 2004.