Type-Aware Web Service Composition Using Boolean Satisfiability Solver

Wonhong Nam Hyunyoung Kil Dongwon Lee*
The Pennsylvania State University, University Park, PA 16802, USA

E-mail: {wnam, hkil, dongwon}@psu.edu

Abstract

The goal of the Web Service Composition (WSC) problem is to find an optimal "composition" of web services to satisfy a given request using their syntactic and/or semantic features, when no single service satisfies it. In this paper, in particular, we study the WSC problem from semantic aspects, exploiting the supertype-subtype relationship among parameters, and propose a novel solution based on the boolean satisfiability problem (SAT). Given a set of web service descriptions and a requirement web service, we reduce the WSC problem into a reachability problem on a state-transition system, and show that the shortest path found is amount to the optimal composition. A preliminary experiment using 7 examples reveals that our proposal can find optimal compositions of web services efficiently.

1. Introduction

Web services are software systems designed to support machine to machine inter-operations over internet. Recently, many researches have been carried out for the web service standard, and these efforts significantly have improved flexible and dynamic functionality of *service oriented architectures* in the current semantic web services. However, a number of research challenges still remain; e.g., automatic web service discovery, web service composition and formal verification for composed web services. Given a set of available web services and a user request, a *web service discovery problem* is to automatically find a web service satisfying the request. Often, the client request cannot, however, be fulfilled by a single pre-existing service. In this case, one desires *web service composition (WSC)* which combines some from a given set of web services to satisfy the requirement.

In this paper, we propose a novel technique to find an *optimal* composition. Given a set of web services and a requirement web service described in WSDL, our algorithm identifies the *shortest sequence of web services* such that we can legally invoke the next web service in each step and achieve the desired requirement eventually. We first reduce the composition problem into a *reachability problem* on a *state-transition system* where the shortest path from the initial state to a goal state corresponds to the shortest sequence of web services. To solve the reachability problem, we employ a state-of-the-art SAT solver, SATZILLA [13]. An important property of our algorithm is that the number of SAT solver invocations required is logarithmic in the size of the web service set. To the best of our knowledge, there is no work to employ SAT solvers for this problem setting even though several approaches [9, 12] use AI planning techniques for web service composition. We report on a preliminary implementation and experiment for our solution, which demonstrate that our technique finds optimal compositions for 7 sample examples for the WSC'07 competition [4] in 1 minute for each.

2. Type-Aware Web Service Composition

In this section, we formalize the notion of web services and their composition we consider in this paper. A *web service* is a tuple $w = (I, O)$ where

- I is a finite set of *input parameters* for w.
- O is a finite set of *output parameters* for w; each input/output parameter $p \in I \cup O$ has a type t_p.

When a web service w is invoked with all the input parameters $i \in I$ with the type t_i, it returns all the output parameters $o \in O$ with the type t_o. Given two types t_1 and t_2, t_1 is a *subtype* of t_2 (denoted by $t_1 <: t_2$) if t_1 is more informative than t_2 so that t_1 can substitute for t_2 everywhere. In this case, t_2 is a *supertype* of t_1. This relation is reflexive (i.e., $t <: t$ for any type t) and transitive (i.e., if $t_1 <: t_2$ and $t_2 <: t_3$ then $t_1 <: t_3$). We assume that the type hierarchy is given; e.g. specified in OWL. Given two web services $w_1(I_1, O_1)$ and $w_2(I_2, O_2)$, we denote $w_1 \sqsupseteq_I w_2$ if w_2 requires less informative inputs than w_1; i.e., for every $i_2 \in I_2$ there exists $i_1 \in I_1$ such that $t_{i_1} <: t_{i_2}$. Given two web services $w_1(I_1, O_1)$ and $w_2(I_2, O_2)$, we denote $w_1 \sqsubseteq_O w_2$ if

*Partially supported by IBM and Microsoft gifts.

w_2 provides more informative outputs than w_1; i.e., for every $o_1 \in O_1$ there exists $o_2 \in O_2$ such that $t_{o_2} <: t_{o_1}$. A *Web service discovery problem* is, given a set W of available web services and a request web service w_r, to find a web service $w \in W$ such that $w_r \sqsupseteq_I w$ and $w_r \sqsubseteq_O w$.

However, it might happen that there is no single web service satisfying the requirement. In that case, we want to find a sequence $w_1 \cdots w_n$ of web services such that we can invoke the next web service in each step and achieve the desired requirement eventually. Formally, we extend the relations, $\sqsupseteq_I$ and $\sqsubseteq_O$, to a sequence of web services as follows.

- $w \sqsupseteq_I w_1 \cdots w_n$ (where $w = (I, O)$ and each $w_j = (I_j, O_j)$) if $\forall 1 \le j \le n$: for every $i_2 \in I_j$ there exists $i_1 \in I \cup \bigcup_{k<j} O_k$ such that $t_{i_1} <: t_{i_2}$.
- $w \sqsubseteq_O w_1 \cdots w_n$ (where $w = (I, O)$ and each $w_j = (I_j, O_j)$) if for every $o_1 \in O$ there exists $o_2 \in \bigcup_{1 \le j \le n} O_j$ such that $t_{o_2} <: t_{o_1}$.

Finally, given a set of available web services W and a service request w_r, a *type-aware web service composition problem* $WC = \langle W, w_r \rangle$ we focus on in this paper is to find a sequence $w_1 \cdots w_n$ (every $w_j \in W$) of web services such that $w_r \sqsupseteq_I w_1 \cdots w_n$ and $w_r \sqsubseteq_O w_1 \cdots w_n$. The optimal solution for this problem is to find a sequence with the minimum value for n.

3. Solving with SAT Solver

To solve a web service composition problem with a boolean satisfiability solver, we first explain how this problem can be reduced into a reachability problem on a state-transition system. Then, we present our encoding to a CNF (Conjunctive Normal Form) formula which is true if and only if there exists a path of length k from an initial state to a goal state of the state-transition system. Finally, we propose our algorithm to find an optimal solution for the problem.

3.1 Reduction to reachability problem

Given a type-aware web service composition problem $WC = \langle W, w_r \rangle$, the problem can be reduced into a reachability problem on a state-transition system. A *state-transition system* is a tuple $S = (X, \Sigma, T)$ where

- X is a finite set of *boolean variables*; a *state* q of S is a valuation for all the variables in X.
- Σ is a set of *input symbols*.
- $T(X, \Sigma, X')$ is a *transition predicate* over $X \cup \Sigma \cup X'$. For a set X of variables, we denote the set of primed variables of X as $X' = \{x' \mid x \in X\}$, which represents a set of variables encoding the successor states. $T(q, a, q')$ is *true* iff q' can be the next state when the input $a \in \Sigma$ is received at the state q.

Given a set $W = \{w_1, \cdots, w_n\}$ of web services where for each j, $w_j = (I_j, O_j)$, we denote as TP a set of types t such that there exists $p \in \bigcup(I_j \cup O_j)$ and t is the type of p. Then, we can construct a state-transition system $S = (X, \Sigma, T)$ corresponding with W as follows:

- $X = \{x_1, \cdots, x_m\}$ where $m = |TP|$; each boolean variable x_j represents whether we have an instance with the type $t_j \in T$ at a state.
- $\Sigma = W$.
- For each j, $T(q, w_j, q') = $ *true* where $q = (b_1, \cdots, b_m)$, $q' = (c_1, \cdots, c_m)$ (each b_k and c_k are *true* or *false*), and $w_j = (I_j, O_j)$ iff (1) for every $i \in I_j$, there exists b_k in q such that b_k is *true* and its corresponding type t_{x_k} is a subtype of the type of i (i.e., $t_{x_k} <: t_i$), (2) if b_l is *true*, c_l is also *true*, and (3) $\forall o \in O_j$: for every variable c_k in q', if its corresponding type t_{x_k} is a supertype of t_o, c_k is *true*. Intuitively, if a web service w_j is invoked at a state q where we have data instances being more informative than inputs of w_j, we proceed to a state q' where we retain all the data instances from q and acquire outputs of w_j as well as their supertypes.

In addition, from a given requirement web service $w_r = (I_{w_r}, O_{w_r})$, we encode an *initial state predicate* $Init(X)$ and a *goal state predicate* $G(X)$ as follows:

- $Init(q) = $ *true* where $q = (b_1, \cdots, b_m)$ iff $\forall i \in I_{w_r}$: for every variable b_j in q, if its type t_{x_j} is a supertype of t_i (i.e., $t_i <: t_{x_j}$), b_j is *true*.
- $G(q) = $ *true* where $q = (b_1, \cdots, b_m)$ iff for every output parameter $o \in O_{w_r}$, there exists b_j in q such that b_j is *true* and its type t_{x_j} is a subtype of t_o (i.e., $t_{x_j} <: t_o$).

Intuitively, we have an initial state where we possess all the data instances corresponding to the input of w_r as well as one corresponding to their supertypes. As goal states, if a state is more informative than the outputs of w_r, it is a goal state. Finally, given a type-aware web service composition problem $WC = \langle W, w_r \rangle$, we can reduce WC into a reachability problem $R = \langle S, Init, G \rangle$ where the shortest path from an initial state to a goal state corresponds to the shortest sequence of web services. We omit a formal proof for our reduction due to space limitation.

3.2 Encoding to CNF formula

Now, we study how to construct a formula $[[R]]_k$ which is true if and only if there exists a path $q_0 \cdots q_k$ of length k for a given reachability problem $R = \langle S, Init, G \rangle$. The formula $[[R]]_k$ is over sets $X_0, \cdots, X_k$ of variables and $W_1, \cdots, W_k$ where each X_j represents a state along the path and W_j encodes a web service invoked in each step. It

Algorithm 1: WebServiceCompositionLinear

```
Input  : a set $W$ of web services and a web service $w_r$.
Output: a sequence of web services.
1 $(S, Init, G) := ReduceToReachabilityProblem(W, w_r)$;
2 for $(k := 0; k \leq |W|; k := k + 1)$ do
3     $f := ConstructCNF(S, Init, G, k)$;
4     if $((path := \textsc{Sat}(f)) \neq null)$ then
5         return $ExtractWSSequence(path)$;
```

essentially represents constraints on $q_0 \cdots q_k$ and $w_1 \cdots w_k$ such that $[[R]]_k$ is satisfiable if and only if q_0 is the initial state, each q_j evolves according to the transition predicate for w_j, and q_k reaches to a goal state. Formally, the formula $[[R]]_k$ is as follows:

$$[[R]]_k \equiv Init(X_0) \wedge \bigwedge_{0 \leq j < k} T(X_j, W_{j+1}, X'_{j+1}) \wedge G(X_k)$$

Since each X_j is a finite set of boolean variables, Σ and W_j are finite, and *Init*, T and G are predicates, we can easily translate $[[R]]_k$ into a CNF formula which is the standard input format for conventional SAT solvers.

3.3 Algorithm for the optimal solution

Since we can use a SAT solver with $[[R]]_k$ to check whether there exists a path of length k from the initial state to a goal state, we are able to find a shortest path simply by increasing the value k from 0 to $|W|$. In the worst case, we check the formula until only $|W|$ as k since multiple executions of any $w \in W$ do not provide more data instances than a single execution of w.

Algorithm 1 presents the linear version. Given a set W of web services and a requirement web service w_r, the algorithm first reduces them into a state-transition system, and initial and goal predicates as Section 3.1 (line 1), and it begins with 1 as the value of k. For each loop, it constructs a CNF formula for k as Section 3.2 (line 3), and checks it with an off-the-shelf SAT solver (line 4). If the formula is satisfiable, the SAT solver returns a truth assignment; otherwise, it returns *null*. Once the algorithm finds a path of the length k, it extracts a web service sequence from the path, and returns the sequence (line 5).

However, we can improve our algorithm from this linear algorithm based on Proposition 1.

Proposition 1. *When there does not exist a path of length k from the initial state to a goal state, there does not exist a path of length $j < k$ from the initial state to a goal state.* □

PROOF. (By contradiction) In our problem setting, if there exists such a path π of length $j < k$ to a goal, then we also have a path π' of length k to the goal by invoking any web service w after π since the invocation of w does not lose any data instance already acquired. (Q.E.D)

Algorithm 2: WebServiceCompositionLogarithmic

```
Input  : a set $W$ of web services and a web service $w_r$.
Output: a sequence of web services.
1 $(S, Init, G) := ReduceToReachabilityProblem(W, w_r)$;
2 $low := 0$;
3 $high := |W|$;
4 $path := null$;
5 $k := pivot$;    /* $pivot$ is predefined. */
6 while $(low \leq high)$ do
7     $f := ConstructCNF(S, Init, G, k)$;
8     if $((tmp := \textsc{Sat}(f)) \neq null)$ then
9         $high := k - 1$;
10        $path := tmp$;
11    else $low := k + 1$;
12    $k := (high + low)/2$;
13 return $ExtractWSSequence(path)$;
```

Our algorithm begins with a pivot value as k. If we find a path of length k, then we again execute the SAT solver with $[[R]]_{(low+k)/2}$ to check whether there exists a shorter path. Otherwise, we retry with $[[R]]_{(k+high)/2}$ as *binary search algorithms*. Using this manner, we can quickly converge to the shortest path. Algorithm 2 presents our logarithmic algorithm to find an optimal solution by only $\log|W|$ executions of a SAT solver which is the most expensive operation in our algorithm. Our algorithm first reduces W and w_r into a state-transition system, and initial and goal predicates (line 1), and it begins with a predefined pivot value as k (line 5). We repeat lines 6–12 until our binary search is completed. In each loop, we construct a CNF formula for k, and check it with a SAT solver. Once we find a path of length k, we again search a shorter path between *low* and $k - 1$. If not, we try to find a path between $k + 1$ and *high*. Finally, after completing the loop, our algorithm extracts a web service sequence from the shortest path we have found, and returns it (line 13).

4. Preliminary Experiment

We have implemented an automatic tool for the logarithmic algorithm in Section 3.3. Given a type hierarchy in a OWL file, and a set of available web services and a query web service in WSDL files, our tool generates a web service sequence in WSBPEL to satisfy the request. Since public web service networks have the small world property [8], we have selected a smaller value for the pivot than the median (we use 10 as the pivot value). To demonstrate that our tools efficiently identify an optimal solution, we have

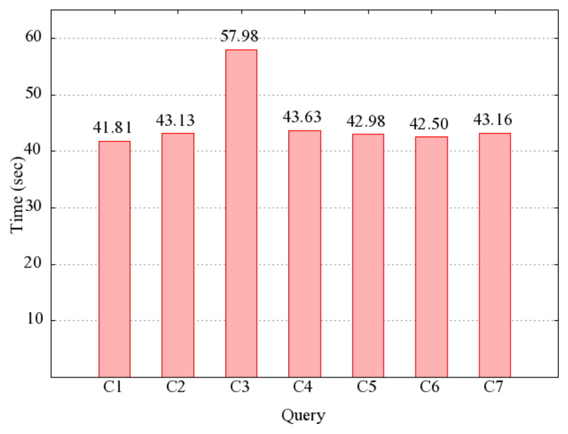

Figure 1. Preliminary experimental result

experimented on the sample example for WSC'07 competition [4] which includes 413 web services and 7 queries. For an off-the-shelf SAT solver, we employ SATZILLA [13] which achieved promising success in SAT 2007 competition [2]. All experiments have been performed on a PC using a 2.4GHz Pentium processor, 2GB memory and a Linux operating system. Figure 1 presents the total execution time[1] in seconds for our preliminary experiment of our logarithmic algorithm.

5. Related Work

A boolean satisfiability (SAT) problem is a well-known NP-complete problem for which there does not exist a scalable solution. Recently, however, many engineering efforts have achieved promising results for a moderately large size of problems [14, 2, 13]. In addition, in every two years the SAT competition [2] is organized to improve the contemporary techniques. Based on this success, there are several researches using SAT techniques to exceed the current limit in their fields [7, 3, 1]. However, there is no study to employ SAT techniques for semantic web service composition. Kautz and Selman [7] adopted SAT techniques for automated planning. In symbolic model checking, [3] proposed a new technique to search a counter-example of length k using a SAT solver. Even though this technique does not provide completeness, it has dramatically increased the bound of models we can analyze. To efficiently solve various games, Alur at el. [1] proposed several techniques using SAT and QBF solvers and compared them.

For web service composition problems, we have a previous work [10, 11] with efficient heuristics such as forward/regression search. However, this work is a new approach using various heuristic techniques implemented in a state-of-the-art SAT solver to explore state space.

[1]We exclude the time for reduction to the state-transition system as the policy of WSC'08 does not include the bootstrap time in the total execution time.

6. Conclusion and Future Work

In this paper, we proposed a novel solution that finds the shortest sequence of web services while respecting types of web service. To identify the optimal solution, our proposal uses a binary search with a boolean satisfiability solver. A preliminary experiment reveals promising results where the tool finds the shortest sequence with logarithmic number of invocations of a SAT solver.

Ample directions are ahead for future work. First, our proposal needs to be extended further to consider not only types of parameters but also true ontologies (in OWL) and reasoning therein. Second, while our implementation uses SAT-based state-space exploration, note that it permits to use other model checking strategies such as BDD-based model checking [6] and counter-example guided abstraction refinement [5]. Thus, we plan to investigate the impact of other model checking strategies in solving the WSC problem.

References

[1] R. Alur, P. Madhusudan, and W. Nam. Symbolic computational techniques for solving games. *STTT*, 7(2):118–128, 2005.

[2] D. Berre, O. Roussel, and L. Simon. The International SAT Competitions. `http://www.satcompetition.org/`.

[3] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, pages 193–207, 1999.

[4] M. B. Blake, W. K.-W. Cheung, M. C. Jaeger, and A. Wombacher. WSC-07: Evolving the web services challenge. In *CEC/EEE*, pages 505–508, 2007.

[5] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, pages 154–169, 2000.

[6] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2000.

[7] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI*, pages 1194–1201, 1996.

[8] H. Kil, S.-C. Oh, and D. Lee. On the topological landscape of web services matchmaking. In *VLDB Workshop on Semantic Matchmaking and Resource Retrieval*, 2006.

[9] B. Srivastava and J. Koehler. Web service composition: Current solutions and open problems. In *ICAPS Workshop on Planning for Web Services*, pages 28–35, 2003.

[10] S. Oh, D. Lee, and S.R.T. Kumara. Web service planner (WSPR): an effective and scalable web service composition algorithm. *JWSR*, 4(1):1–23, 2007.

[11] S. Oh, J. Yoo, H. Kil, D. Lee, and S.R.T. Kumara. Semantic web-service discovery and composition using flexible parameter matching. In *CEC/EEE*, pages 533–542, 2007.

[12] E. Sirin, B. Parsia, D. Wu, J.A. Hendler, and D.S. Nau. HTN planning for web service composition using SHOP2. *J. Web Sem.*, 1(4):377–396, 2004.

[13] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla-07: The design and analysis of an algorithm portfolio for SAT. In *CP*, pages 712–727, 2007.

[14] L. Zhang and S. Malik. The quest for efficient Boolean satisfiability solvers. In *CAV*, pages 17–36, 2002.