Towards Scalable Web Service Composition With Partial Matches

Adina Sirbu, Jörg Hoffmann
Semantic Technology Institute (STI) Innsbruck
University of Innsbruck, Austria
{adina.sirbu, joerg.hoffmann}@sti2.at

Abstract

We investigate scalable algorithms for automated composition (WSC) of Semantic Web Services. Our notion of WSC is very general: the composition semantics includes background knowledge and we use the most general notion of matching, partial matches*, where several web services can cooperate, each covering only a part of a requirement. Unsurprisingly, automatic composition in this setting is very hard. We identify a special case with simpler semantics, which covers many relevant scenarios. We develop a composition tool for this special case. Our goal is to achieve scalability: we overcome large search spaces by guiding the search using heuristic techniques. The computed solutions are optimal up to a constant factor. We test our approach on a simple, yet powerful real world use-case; the initial results attest the potential of the approach.*

1 Introduction

Automatic processing of Semantic Web Services (SWS) is the main advantage of combining Semantic Web and Web service technologies. In this paper we concentrate on one important challenge, that of automatically composing SWS (WSC). Through WSC, existing web services can be reused, thus reducing development time and effort. We address composition at the functional level, where only the overall behavior of web services is taken into account. This corresponds to the WSMO [6] web service capability and OWL-S [4] service profile level. The input/output behavior of web services is mapped to input/output parameters on which preconditions and effects are specified.

In any task that involves SWS one needs support for background knowledge, modeled as ontologies. In the case of WSC, almost all existing solutions compile the problem into AI Planning formalisms. The motivation is that planning tools have become many times more scalable in recent years. The problem here is that those tools cannot handle background ontologies.

Background theories are explored in some planning works, e.g., [5]. However, incorporating this notion into modern scalable planning tools poses serious challenges, and has not even been tried. Due to the background theory, even computing a state transition (now a form of belief revision) is a computationally very hard task. The existing planning tools dealing with background theories map the problem into generic deduction, which is well known for its lack of scalability. As a result, many existing planning tools dealing with WSC, e.g., [12, 1], ignore the ontology and assume *exact matches*. Few approaches allow restricted ontologies and assume *plug-in matches*, e.g., [9]. In contrast, we use the most general notion of matching, *partial matches*, where several web services can cooperate, each covering only a part of a requirement.

An interesting special case of WSC has been recently identified in [7]. Web services may output new constants (the outputs model generated data). Now, if all ramifications of a web service concern only propositions involving at least one new constant, then a belief revision is not necessary. This special case has been termed *forward effects*: the effects are "forward" in the sense that no backwards-directed belief revision is necessary. Many WSC scenarios from literature and real case studies have forward effects, involving the creation of tickets or confirmations.

The work reported in this paper naturally builds on [7]. However, while [7] focuses on a restriction of forward effects which can be compiled into a planning formalism, here we develop a dedicated tool for WSC with forward effects. By solving the problem in its natural form we expect to obtain a more efficient tool, optimized for problems that are composition-specific. For example, in comparison to the planning tasks which involve a relatively small number of predefined actions, WSC has to deal with a large number of web services, and therefore with a huge branching factor.

We present the formalism for WSC with partial matches. As shown in [7], testing if a sequence of web service applications is a solution (*solution testing*) is Π_2^p-complete in general, but only **coNP**-complete with forward effects. Although **coNP** is still hard, *planning under uncertainty* has

the same complexity of solution testing and scalable tools have been developed. Our algorithms are based on ideas underlying one such tool, namely Conformant-FF [8]. These algorithms are presented in detail. We define the search space by encoding forward effects beliefs as propositional CNFs. To deal with large search spaces, we control the search via heuristic techniques. We design an admissible heuristic function, and achieve optimality up to a constant factor by combining it with a weighted A* search strategy.

The paper is structured as follows. Section 2 introduces our reference example. We present the formalism and discuss the forward effects in section 3. In section 4 we build a tool for forward effects. Section 5 presents our initial results. Section 6 positions the work in a broader context. We conclude and discuss future directions in section 7.

2 A Motivating Example

Our reference example (Fig. 1) is a variation of the travel scenario more widely known as the "virtual travel agency" (VTA). The web services in this scenario serve for making travel arrangements. They have forward effects because they generate new constants corresponding to tickets and reservations, and set their properties relative to the inputs.

The general user request is to set up a travel itinerary: flight, shuttle service from/to airport, accommodation. This request can be further refined when concrete input data is available, for example when the destination is known. A solution to such a composition task is a subset of the available web services and an appropriate ordering, such that the subset completely satisfies the request. A general request is valuable when we wish to solve the composition task at design-time. Through composition, we search for the subset of web services that can theoretically fulfill all concrete user requests. Then, a solution does not need to be recomputed when real data is available, it can simply be obtained by removing from the general solution the web services that are always non-applicable.

To illustrate the importance of *partial matches*, consider the following scenario fragment. The ontology defines the concepts $AccomReq$, $HotelReq$, $B\&BReq$ and $HostelReq$, where $HotelReq$, $B\&BReq$ and $HostelReq$ are more specific then $AccomReq$ (i.e., $HotelReq \subseteq AccomReq$). The ontology further states that $HotelReq \cup B\&BReq \cup HostelReq \supseteq AccomReq$, that is, the $AccomReq$ is completely covered by its sub-concepts.

Our service pool contains several web services that require a booking request and provide a booking confirmation. However, none of them deals with a general $AccomReq$. The "backpackers.com" web service handles hostel accommodation, "venere.com" deals only with hotels, while B&Bs are covered by "bedandbreakfast.com". A solution to this problem therefore needs to apply the three

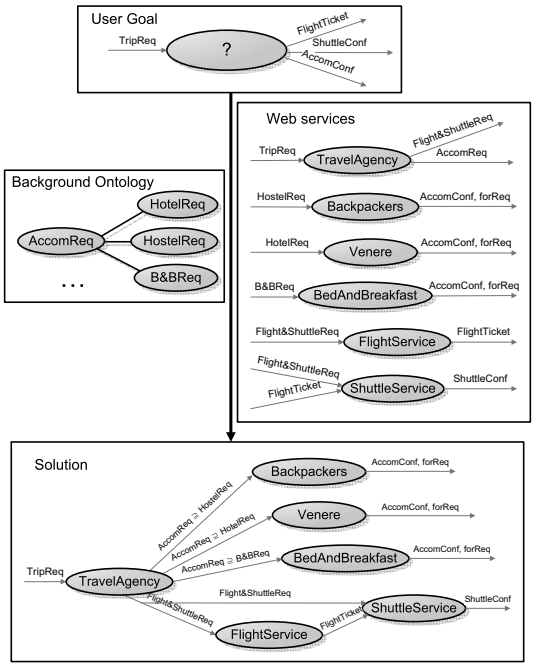

Figure 1: Illustration of Example

distinct web services in conjunction.

In this example, reasoning over the background ontology is necessary to (1) understand which services can be used, and (2) test whether a given composition is actually a solution. Enforcing *exact matches* would require $AccomReq = HotelReq$ instead of $AccomReq \cap HotelReq \neq \emptyset$. Obviously, this renders the example unsolvable. If *plug-in* matches are allowed, we would require $AccomReq \subseteq HotelReq$ rather than $AccomReq = HotelReq$. This still cannot solve our example.

3 WSC with Partial Matches

We introduce the syntax and semantics of the WSC formalism. Further, we define and analyze the forward effects special case. We refer to [7] for details and proofs of the formal claims in this section.

3.1 WSC Formalism

We assume supplies of logical predicates p, q, variable names x, y and constant names a, b, c; *(ground) literals* are defined as usual. For variables X, $\mathcal{L}^X$ is the set of literals using only variables from X. We write $l[X]$ for a literal l with variable arguments X. For a tuple C of constants substituting X, we write $l[C/X]$. In the same way, we use the substitution notation for any construct involving variables. Positive ground literals are *propositions*.

A *clause* is a disjunction of literals with universal quantification on the outside, e.g. $\forall x.(\neg p(x) \vee q(x))$. An *ontology* is a conjunction of clauses. An *web service* w is

a tuple $(X_w, \text{pre}_w, Y_w, \text{eff}_w)$, where X_w, Y_w are sets of variables, pre_w is a conjunction of literals from $\mathcal{L}^{X_w}$, and eff_w is a conjunction of literals from $\mathcal{L}^{X_w \cup Y_w}$. The intended meaning is that X_w are the inputs and Y_w the outputs, i.e., the new constants created by the web service. For a web service w, a web service *application* a is given by $(\text{pre}_a, \text{eff}_a) \equiv (\text{pre}_w, \text{eff}_w)[C_a/X_w, E_a/Y_w]$ where C_a and E_a are vectors of constants; for E_a we require that the constants are pairwise different.

$\mathcal{WSC}$ tasks are tuples $(\mathcal{P}, \mathcal{O}, \mathcal{W}, \mathcal{U})$. Here, $\mathcal{P}$ are predicates; $\mathcal{O}$ is the ontology; $\mathcal{W}$ is the set of web services; $\mathcal{U}$ is the user requirement, defined as a triple $(C^p_{\mathcal{U}}, \phi^p_{\mathcal{U}}, \phi^e_{\mathcal{U}})$. $\phi^p_{\mathcal{U}}$ and $\phi^e_{\mathcal{U}}$ are conjunctions of literals corresponding to the user requirement precondition and effect, respectively; $C^p_{\mathcal{U}}$ is a set of constants corresponding to the variables in $\phi^p_{\mathcal{U}}$. The interpretation will be that the constants $C^p_{\mathcal{U}}$ satisfy $\phi^p_{\mathcal{U}}$, i.e., these variables become "generic constants" about which we know only the user requirement precondition. We want to apply web services to reach a situation where an appropriate instantiation of the variables in $\phi^e_{\mathcal{U}}$ is guaranteed to exist.

Example. *We formalize the scenario introduced in section 2:*

- $\mathcal{P}$ *includes* $\{TripReq, AccomReq, HostelReq, HotelReq, B\&BReq, Flight\&ShuttleReq, AccomConf, forReq, FlightTicket, ShuttleConf\}$
- $\mathcal{O}$ *is:* $\forall x.(\neg HotelReq(x) \vee AccomReq(x))$ *and* $\forall x.(\neg HostelReq(x) \vee AccomReq(x))$ *and* $\forall x.(\neg B\&BReq(x) \vee AccomReq(x))$ *and* $\forall x.(\neg AccomReq(x) \vee HotelReq(x) \vee HostelReq(x) \vee B\&BReq(x))$
- $\mathcal{W}$ *includes:* $w_{TravelAgency} = (\{x\}, TripReq(x), \{y, z\}, Flight\&ShuttleReq(y) \wedge AccomReq(z))$
  $w_{Backpackers} = (\{x\}, HostelReq(x), \{y\}, AccomConf(y) \wedge forReq(y, x))$
  $w_{Venere} = (\{x\}, HotelReq(x), \{y\}, AccomConf(y) \wedge forReq(y, x))$
  $w_{BedAndBreakfast} = (\{x\}, B\&BReq(x), \{y\}, AccomConf(y) \wedge forReq(y, x))$
- $\mathcal{U}$ *is:* $(\{icws'08\}, TripReq(x), \exists y, z, t.(FlightTicket(y) \wedge ShuttleConf(z) \wedge AccomConf(t)))$

Assume we are given a task $(\mathcal{P}, \mathcal{O}, \mathcal{W}, \mathcal{U})$. *States* in our formalism are pairs (C_s, I_s) where C_s is a set of constants, and I_s is a *C_s-interpretation*, i.e., an interpretation of the predicates $\mathcal{P}$ over the constants C_s. Given a state s and a web service application a, a is *applicable in* s if $I_s \models \text{pre}_a$, $C_a \subseteq C_s$, and $E_a \cap C_s = \emptyset$. We allow *parallel* applications. These are sets of applications performed at the same point in time. The result of a parallel application A in a state s is

$$res(s, A) := \{(C', I') \mid C' = C_s \cup \bigcup_{a \in A, appl(s,a)} E_a, \\ I' \in min(s, C', \mathcal{O} \wedge \bigwedge_{a \in A, appl(s,a)} \text{eff}_a)\}$$

Here, $min(s, C', \phi)$ is the set of all C'-interpretations that satisfy ϕ and that are minimal with respect to the partial order defined by $I_1 \leq I_2$:iff for all propositions p over C_s, if $I_2(p) = I_s(p)$ then $I_1(p) = I_s(p)$. This is a standard semantics where the ramification problem is addressed by requiring minimal changes to the predecessor state s [17]. A is *inconsistent* with s iff $res(s, A) = \emptyset$; this can happen in case of conflicts. Note that in the definition of $res(s, A)$, non-applicable $a \in A$ are ignored; in particular they are allowed. This realizes partial matches: a can be applied as soon as it matches at least one possible situation.

We refer to the set of states possible at a given time as a *belief*. The *initial belief* is $b_0 := \{s \mid C_s = C^p_{\mathcal{U}}, s \models \mathcal{O} \wedge \phi^p_{\mathcal{U}}\}$. A is inconsistent with a belief b if it is inconsistent with at least one $s \in b$. In this case, $res(b, A)$ is undefined; else, it is $\bigcup_{s \in b} res(s, A)$. This is extended to sequences of parallel applications in the obvious way. A *solution* is a sequence $\langle A_1, \ldots, A_n \rangle$ s.t. for all $s \in res(b_0, \langle A_1, \ldots, A_n \rangle) : s \models \phi^e_{\mathcal{U}}$.

When assuming *fixed arity* – a constant upper bound on the arity of all variable vectors (e.g., used in predicates) – transformation to a propositional representation is polynomial. Even in this case, solution testing is Π^p_2-complete in $\mathcal{WSC}$. Further, polynomially bounded solution existence has been proved to be Σ^p_3-complete.

3.2 Forward Effects

The high complexity of $\mathcal{WSC}$ motivates the search for interesting special cases. We consider a case where every change an application makes to a state involves a new constant. A $\mathcal{WSC}$ task $(\mathcal{P}, \mathcal{O}, \mathcal{W}, \mathcal{U})$ has *forward effects* iff:

- For all $w \in \mathcal{W}$, and for all $l[X] \in \text{eff}_w$, we have $X \cap Y_w \neq \emptyset$. In words, the variables of every effect literal contain at least one output variable.
- For all clauses $cl[X] \in \mathcal{O}$, where $cl[X] = \forall X.(l_1[X_1] \vee \cdots \vee l_n[X_n])$, we have $X = X_1 = \cdots = X_n$. In words, in every clause all literals share the same arguments.

In our example, w_{Venere} creates a $AccomConf$ and relates it to the input constant of type $HotelReq$. This does not affect the previous knowledge about the input constant, because of the second condition: effects involving new constants can only affect literals involving new constants.

The set of all such tasks is denoted with $\mathcal{WSC}|_{fwd}$.

Given a state s and a parallel application A, define

$$res|_{fwd}(s, A) := \{(C', I') \mid C' = C_s \cup \bigcup_{a \in A, exec(s,a)} E_a, \\ I'|_{C_s} = I_s, I' \models \mathcal{O} \wedge \bigwedge_{a \in A, exec(s,a)} \text{eff}_a\}$$

Here $I'|_{C_s}$ is the restriction of I' to propositions over C_s.

Proposition 3.1 (Semantics of $\mathcal{WSC}|_{fwd}$)**.** *Assume a $\mathcal{WSC}|_{fwd}$ task, a state s, and a parallel application A. Then $res(s, A) = res|_{fwd}(s, A)$.*

Hence, the semantics of applications becomes a lot simpler with forward effects, no longer needing the notion of minimal changes with respect to the previous state. In difference to $\mathcal{WSC}$, we can make sure that no inconsistent applications will occur by filtering out certain applications in a

preprocessing step. If A is inconsistent with a belief b, then there exists $a \in A$ s.t. $\mathcal{O} \wedge \text{eff}_a$ is unsatisfiable. a is called *contradictory*. Obviously, a contradictory application will never yield a successor state; it can be filtered out prior to composition, without affecting solution existence. Thanks to this, and thanks to the simpler semantics as per Proposition 3.1, solution testing is much easier in $\mathcal{WSC}|_{fwd}$ than in $\mathcal{WSC}$: it has been proved to be **coNP**-complete. Also, polynomially bounded solution existence has been proved to be Σ_2^p-complete.

4 From Conformant-FF to Forward Effects

We develop a tool for $\mathcal{WSC}|_{fwd}$ by using the correspondence between WSC with partial matches and planning under uncertainty. The latter is the task of generating plans given uncertainty about the initial state, and without any sensing capabilities during plan execution. This correspondence is as follows: in both cases the world is described in terms of a formula and the composed solution should work for every situation satisfying that formula.

Among the tools for planning under uncertainty, of particular importance here is Conformant-FF [8]. Planning under uncertainty can be transformed into a search problem in belief space, the space whose elements are sets of possible worlds. The key observation in Conformant-FF is that, for a setting where the goal and action preconditions are reduced to simple conjunctions of propositions, it is sufficient to know the propositions that are true in the intersection of worlds contained in a belief state. Based on this observation, Conformant-FF trades space for time. Instead of maintaining complete knowledge of a belief state s in memory, the planner maintains for s only the known propositions and the path leading to it (the initial belief state and the action sequence that leads to s). To check for fulfillment of the goal and action preconditions, the planner checks that every proposition is implied by a CNF formula that captures the semantics of the action sequence. This approach can be naturally extended to WSC with forward effects, since forward effects beliefs can be represented as CNFs. A similar encoding is not feasible in the general case, because the compiled formulas may be exponentially large.

Making such an approach efficient is a challenging task. Preconditions and effects are more complex in our case, making it difficult to apply the optimizations used in Conformant-FF, for example caching and reusing results. Moreover, Conformant-FF (and every existing planning tool) does not allow the generation of new constants. It is critical to devise heuristics for identifying which constants are important, since exponentially many constants may be generated in general.

In section 4.1 we generate formulas that encode the semantics of forward effects beliefs. We present our heuristic function in section 4.2. Due to limited space, many details and proofs of the formal claims in this section have been omitted. They are available in a technical report [14].

4.1 Adapting the Generation of Formulas

Assume a $\mathcal{WSC}|_{fwd}$ task $(\mathcal{P}, \mathcal{O}, \mathcal{W}, \mathcal{U})$ with fixed arity and without contradictory applications and a sequence $\overline{A} = \langle A_1, \ldots, A_n \rangle$ of parallel applications. Assume also that any two applications $a, a' \in \overline{A}$ are compatible, i.e. either $E_a \cap E_{a'} = \emptyset$, or $\text{eff}_a = \text{eff}_{a'}$. In order to determine the known and negatively known propositions in a belief state, and be able to test the satisfiability of the goal formula, we construct a CNF corresponding to the semantics of $\overline{A}$.

We introduce a predicate $exists(c, t)$, useful for keeping track of the constants generated at different points along the execution of $\overline{A}$. Further, in order to keep the number of generated clauses polynomial, we introduce a predicate $appl(a, t)$, completely determined by preconditions and exists literals. Except for these predicates, there is no need for timestamps, because once an application has generated its outputs, the properties remain fixed.

The CNF corresponding to $\overline{A}$ can be computed incrementally. We denote with φ_t, $0 \leq t \leq n$, the formula associated with $\langle A_1, \ldots, A_t \rangle$. Corresponding to the empty sequence, φ_0 contains the instantiation of ontology $\mathcal{O}$ and initial literals $\phi_{\mathcal{U}}^p$ with $C_{\mathcal{U}}^p$, and also $\bigwedge_{c \in C_{\mathcal{U}}^p} exists(c, 0)$.

For all t, $0 \leq t < n$, $\varphi_{t+1} = \varphi_t \wedge \delta_{t+1}$, where δ_{t+1} is a CNF consisting of all clauses related to A_{t+1}. For simplicity, we denote with E_t the set of constants that might exist at t, i.e. $E_t = C_{\mathcal{U}}^p \cup \bigcup_{a \in \langle A_1, \ldots, A_t \rangle} E_a$.

First, δ_{t+1} includes the instantiation of ontology $\mathcal{O}$ with E_{t+1}. As an optimization, we enforce that the tuples contain at least one constant from $\bigcup_{a \in A_{t+1}} E_a$. For all $c \in \bigcup_{a \in A_{t+1}} E_a$ we add a clause $\neg exists(c, 0)$ because all outputs are generated (by construction of applications).

For all $a \in A_{t+1}$, we add clauses defining applicability: $\bigwedge_{l \in \text{pre}_a} l \wedge \bigwedge_{c \in C_a} exists(c, t) \wedge \bigwedge_{e \in E_a} \neg exists(e, t) \iff appl(a, t)$. We then add effect clauses. If a can be applied, eff_a is known to hold, and each $e \in E_a$ is known to exist at $t + 1$: $appl(a, t) \Rightarrow \bigwedge_{l \in \text{eff}_a} l \wedge \bigwedge_{e \in E_a} exists(e, t + 1)$

We continue with frame axioms for the $exists$ predicate. For every $c \in E_{t+1}$, we add a negative frame axiom stating that if c did not exist at t, and no $a \in A_{t+1}$ that could create it was applicable, then it does not exist at $t + 1$: $exists(c, t) \vee \bigvee_{a \in A_{t+1}, c \in E_a} appl(a, t) \vee \neg exists(c, t + 1)$ We add also a positive frame axiom (once created, a constant will exist until step n): $exists(c, t) \Rightarrow exists(c, t+1)$

For the output constants in A_{t+1}, we add frame axioms related to the past. For every $c \in \bigcup_{a \in A_{t+1}} E_a$, and for every timestep $1 \leq i \leq t$, we insert a negative frame axiom: $exists(c, i-1) \vee \bigvee_{a \in A_i, c \in E_a} appl(a, i) \vee \neg exists(c, i)$ and a positive frame axiom: $exists(c, i - 1) \Rightarrow exists(c, i)$

Example. *Consider the following applications:* $a_{TA} = w_{TravelAgency}[\{icws'08\}/\{x\}, \{c_1, c_2\}/\{y, z\}]$ *and* $a_V = w_{Venere}[\{c_2\}/\{x\}, \{c_3\}/\{y\}]$. *We construct a sequence of parallel applications as* $\langle\{a_{TA}\}, \{a_V\}\rangle$. *The effect clauses of* a_V*, here directly as CNFs, are:*

- $\neg appl(a_V, 1) \vee AccomConf(c_3)$
- $\neg appl(a_V, 1) \vee forReq(c_3, c_2)$
- $\neg appl(a_V, 1) \vee exists(c_3, 2)$

Solution Testing. In the following, we clarify the meaning of an execution and the correspondence between executions and our CNF formulas. We then show that testing whether a sequence $\overline{A} = \langle A_1, \ldots, A_n\rangle$ is a solution for a given $\mathcal{WSC}|_{fwd}$ task can be reduced to a single SAT test.

Given a $\mathcal{WSC}|_{fwd}$ task, an execution of $\overline{A}$ is defined by a pair $(\overline{C}, \overline{I})$, where $\overline{C}$ is an array of sets of constants, and $\overline{I}$ is an array of interpretations. Each I_t $(0 \leq t \leq n)$ is an interpretation of the predicates $\mathcal{P}$ over the corresponding set of constants C_t. For all $1 \leq t \leq n$, and $a \in A_t$, a is applicable if $I_{t-1} \models \text{pre}_a, C_a \subseteq C_{t-1}, E_a \cap C_{t-1} = \emptyset$. Then the pair $(\overline{C}, \overline{I})$ has the following properties:

- $(C_0, I_0) \in b_0$, where b_0 is the initial belief
- for all t, $1 \leq t \leq n$,
  - $I_t \models \mathcal{O}$
  - for all $a \in A_t$, if a is applicable, then $I_t \models \text{eff}_a$
  - $C_t = C_{t-1} \cup \{E_a | a \in A_t, a \text{ is applicable}\}$

For all I_i, I_j $(0 \leq i \leq j \leq n)$ interpretations of $\mathcal{P}$ over C_i, respectively C_j, we consider that I_j contains I_i, denoted by $I_i \subseteq I_j$, if $I_j|_{C_i} = I_i$, and $C_i \subseteq C_j$. Then, for any execution of $\overline{A}$ given by $(\overline{C}, \overline{I})$, we have that $I_0 \subseteq \ldots \subseteq I_n$. This follows directly from the definition of a result state in the forward effects: $I_{t+1}|_{C_t} = I_t$, $C_t \subseteq C_{t+1}$, $0 \leq t < n$.

We say that a literal l holds after $\overline{A}$ if, for all $(\overline{C}, \overline{I})$ executions of $\overline{A}$, $I_n \models \exists X.(l[X])$, with the quantifier restricted to C_n. Further, we proved that the following correspondence exists between our formula φ_n and the executions $(\overline{C}, \overline{I})$: for all literals l over $\mathcal{P}$, $\varphi_n \Rightarrow \exists X.(l[X] \wedge \bigwedge_{x \in X} exists(x, n)) \iff l$ holds after $\overline{A}$. We extend these statements to conjunctions of literals. Then, $\overline{A}$ is a solution iff the goal $\phi^e_{\mathcal{U}}$ holds after $\overline{A}$. We can then write our solution test as a single SAT test.

Proposition 4.1 (Solution testing equivalence)**.** *Assume a* $\mathcal{WSC}|_{fwd}$ *task* $(\mathcal{P}, \mathcal{O}, \mathcal{W}, \mathcal{U})$ *with fixed arity and without contradictory applications, and a sequence* $\overline{A} = \langle A_1, \ldots, A_n\rangle$ *of parallel applications, each* A_t *representing a compatible set of applications. Then* $\overline{A}$ *is a solution iff* $\varphi_n \wedge \neg\exists X.(\bigwedge_{l \in \phi^e_{\mathcal{U}}[X]} l \wedge \bigwedge_{x \in X} exists(x, n))$ *is unsatisfiable.*

Optimizations. Both entailment of the goal and applicability can be checked with a single SAT call. Alternatively, these tests can be decomposed for individual propositions. In case of φ_n, we can further determine truth values by unit propagation. The knowledge about propositions can then be used to simplify the formulas; it can also be used when instantiating web services, by pruning applications as soon as one of the precondition literals is known to be false.

An important speed-up in solving a $\mathcal{WSC}|_{fwd}$ task is given by *mutexes*. These are generated based on a directed graph having as nodes the predicates in $\mathcal{P}$, $G := (\mathcal{P}, E)$. For every pair $p, q \in \mathcal{P}$ having the same arity, E contains:

- an edge (p, q) if they appear together in a subsumption clause, where p is parent
- edges (p, q) and (q, p) if they appear together in any clause in $\mathcal{O}$; also, if they occur in positive literals having the same arguments in the effects of a web service $w \in \mathcal{W}$ or in the user requirement precondition $\phi^p_{\mathcal{U}}$.

For every pair $p, q \in \mathcal{P}$ we check for a directed path from p to q or from q to p. Because a concept can be the sub-concept of several concepts, we check if there exists a predicate r such that there is a directed path from both p to r and q to r. If none of these conditions is satisfied, we add to the ontology $\mathcal{O}$ a mutex clause $\forall\overline{X}.(\neg p(\overline{X}) \vee \neg q(\overline{X}))$.

We have proved that completeness is preserved when applying the mutexes. The optimization is therefore to solve the constrained task and further test every found solution.

4.2 Heuristic Function

Practical scenarios are expected to involve large sets of web services, and huge search spaces (of partial compositions). To overcome these huge search spaces, we introduce a heuristic function tailored for the forward effects. Here again, we started from Conformant-FF: for each belief, Conformant-FF computes a relaxed solution using approximate SAT reasoning, and uses it to guide the search. The length of this solution is the heuristic function: an estimate of how much more effort is required to complete the search state. Further, the actions contained in the relaxed solution provide an estimate of which actions are most relevant to complete the search state.

We use the same principle, but apply a different relaxation. In Conformant-FF, the relaxation is to assume that all delete lists are empty. Relaxing applications to have only positive effects is not relevant in our setting, where negative effects can be useful for reaching the goal (either directly, or through consequences). Our relaxation is to ignore logical conflicts and allow applications to output *the same* constants. This is a generalization of ignoring delete lists, which addresses a critical problem of $\mathcal{WSC}$, the fact that in general exponentially many constants may be generated.

Given a belief determined by a sequence of parallel applications $\overline{A}$, we compute the heuristic function with a forward step, and the recommended applications with a backward step. In the forward step we build a relaxed graph which gives an approximation of the number of applications necessary for achieving the goal and the data structures from which a relaxed solution can be extracted. Sim-

ilar to the conformant setting, our relaxed graph is a sequence of alternating literal and application layers.

We first present in detail the data structures. We continue with the basic mechanism for building the relaxed graph, which we also illustrate on our reference example. The actual algorithm, together with the algorithm for building a time step (an application and a literal layer), that for computing the approximate test, as well as that for identifying a relaxed solution can be found in [14].

At any step t, literals in Conformant-FF can be either known or unknown (they are true when starting from some initial world states, and false when starting from others). In our setting, this distinction is not sufficient, and we further distinguish "unknown" literals into partial and decision literals. Literals can therefore be:

- *known*, if they follow from the effects of an application that is always applicable, or if we determine using an approximate test that they may cover all situations;
- *partial*, if they follow from the effects of an application that is partially applicable;
- *decision*, if they appear only in the preconditions of applications.

Neither of these literals needs to be timestamped. It is enough to timestamp the constants: based on its constants, the step at which a literal occurs is easily determined.

The relation between known, partial and decision literals is as follows. A decision literal is linked to any literal that can possibly influence its truth value. These links are used in the approximate test for filtering purposes: only relevant applications will be taken into account. We connect partial literals to decision literals using conditions, which are computed incrementally. For a decision literal, the condition is itself. For a partial literal, the condition is a formula constructed based on the applications that achieve the literal, and the conditions of their preconditions. A partial literal becomes known if its condition is *approximatively* implied by the ontology, the initial literals and the effects of (a subset of) already applied web service applications.

The algorithm for building the relaxed graph proceeds as follows. First, it builds the layers referring to the past. We compute these layers because we do not have complete knowledge of the truth value of literals, the only knowledge we have is that obtained by test decomposition (section 4.1). Also, there may be implications between literals that are relevant for the entire graph. Different from Conformant-FF, we do not relax the applications from the past. The number of constants created at each past step is likely to be small. What we gain is that in the approximated test we can conjunctively add the effects of applications from $-n, \ldots, -1$: there will be no contradictions.

The algorithm continues with the 0+ layers. As long as the goal has not been reached, we generate a set of M constants, where M is the maximum number of outputs over

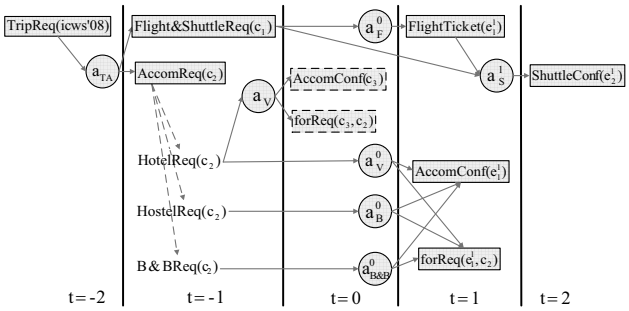

Figure 2: Fragment of the relaxed graph for $\langle\{a_{TA}\},\{a_V\}\rangle$

all web services in $\mathcal{W}$. We then create a set of relaxed applications, allowing them to output the same constants, the newly generated ones. We use these applications to build a new layer. If the new constants behave exactly like the previous ones the goal cannot be reached from the input belief, and we return ∞ such that it can be pruned from the search.

Fig. 2 shows the relaxed graph for the application sequence $\langle\{a_{TA}\},\{a_V\}\rangle$ (introduced in section 4.1). To improve readability, only the applications relevant for the approximate goal test are displayed. It is important to note that all possible applications will be created, including for example for $t = 0$ applications such as $w_{TravelAgency}[\{icws'08\}/\{x\},\{e_1^1,e_2^1\}/\{y,z\}]$. We distinguish between known (gray, solid line boxes), partial (gray, dotted line boxes) and decision literals (without boxes). At $t = 1$, $AccomConf(e_1^1)$ and $forReq(e_1^1,c_1)$ will first be marked as partial. They become known because their condition ($HotelReq(c_2) \vee HostelReq(c_2) \vee B\&BReq(c_2)$) is determined to hold in the approximate test. At $t = 2$, the goal is reached because there exists a constant substitution for the goal formula such that the literals are known. Thus, the heuristic value returned is 2.

We have proved that our heuristic function is admissible, i.e., it never overestimates the number of applications necessary to reach the goal. Therefore, it can be used with search algorithms such as A* to find optimal solutions.

Proposition 4.2 (Admissibility)**.** *Assume a $\mathcal{WSC}|_{fwd}$ task $(\mathcal{P},\mathcal{O},\mathcal{W},\mathcal{U})$ with fixed arity and without contradictory applications and a sequence of parallel applications $\overline{A} = \langle A_{-n},\ldots,A_{-1}\rangle$ for which any two applications are compatible. Assume that $\langle\{a_0\},\ldots,\{a_{m-1}\}\rangle$ is a solution for the belief $b_0 = res(b_{-n},\overline{A})$. Then* build-graph*$(\overline{A},\mathcal{P},\mathcal{O},\mathcal{W},\mathcal{U})$ returns t_{max} such that $t_{max} \leq m$.*

5 Experimental Results

We have implemented our approach into a prototype tool. The tool accepts WSMO background ontologies, web services and goals, all specified in a subset of the WSML language. Reasoning is performed using an integrated SAT solver, namely SAT4J (http://www.sat4j.org). We con-

ducted a series of experiments in order to to establish the benefits of using our heuristic function (section 4.2).

We use for testing the scenario introduced in section 2. Although rather small (6 web services), the scenario is powerful due to the generality of our goal and the high degree of parallelism. Already at the second step, even when introducing mutexes, at least four web service applications can be applied, this leading to a very high branching factor. To observe the behavior of our tool with a larger service pool, we experimented with adding extra web services that do not contribute to the solution. The case likely to occur in practice is that the web services are first filtered by a discovery tool. To simulate this situation, we created extra web services using the set of predicates from the original task.

All tests have been performed using a Dual Core CPU running at 1.83GHz, and allowing at most 1.5GB memory. Because the performance varies dramatically depending on the randomized web services, for each tested situation we have generated 10 sets of web services. Each result is therefore the median value over the 10 runs corresponding to the fixed sets. For each test, we applied a timeout of 30 minutes.

We experimented with multiple configurations. Table 1 displays the results for the configurations using mutexes. First, $\mathcal{W}$ is the number of original plus generated web services. Then, "blind" search uses only the basic optimizations, while "weighted A*" search uses also the heuristic (with a factor of 1 for the path cost and a factor of 3 for the heuristic). Weighted A* uses also a limited lookahead based on the approximate solutions constructed in the heuristic function (the lookahead is discussed in the next experiment). We compare the performance on two criteria: total running time expressed in seconds (T), and number of generated search states (GS). We observe that there is a definite benefit of using the heuristic, according to both criteria.

We discuss now the additional parameter introduced for the configuration using both mutexes and heuristic. In general, web service applications in the relaxed solution may not be applicable in the actual search state. However, in this particular scenario, we have observed that the relaxed solutions are of good quality, and more than just the first application in the solution can be used for guiding the search. The lookahead parameter sets the number of applications from the relaxed solution that should be considered. It is important to note that by using a lookahead greater than 1 we give up optimality. Similar directions have already been explored in the planning community. Investigating this direction in more detail, in our context, is future work.

We experimented with different lookahead values, at the same time increasing the number of web services. Table 2 shows our results for a lookahead of up to 4 (higher values yielded similar results). We observe that when using only the first application from the relaxed plan (lookahead = 1) the runtime is very high, meaning that the time required to compute the heuristic function is very high. We conclude that our heuristic does provide useful guidance, but is still too costly to compute. Not shown in the table is that for a lookahead of up to 3 all the solutions returned are optimal, from which we conclude that for this particular scenario the relaxed solution is reliable for at most 3 steps.

$\mathcal{W}$	blind		weighted A*	
	T	GS	T	GS
6	14.219	120	1.641	20
6+1	14.922	120	1.906	20
6+2	17.454	143	1.829	20
6+3	492.840	754	1.891	20
6+4	>30 min		2.765	26
6+10	>30 min		4.750	41
6+20	>30 min		95.000	190
6+30	>30 min		>30 min	

Table 1: Results when using mutexes

We will try to remedy this problem in two ways. First, our current implementation leaves enough room for improvement. For example, roughly half of the running time is spent performing satisfiability tests. Although large, our CNFs are relatively easy to compute. We will experiment with using a naive standard DPLL solver instead of the fairly complex SAT4J. Also, the formulas are similar, and in many cases, incremental. We will take advantage of this similarity and devise a mechanism for caching results. Secondly, we will experiment with less costly algorithms for computing the heuristic function, for example by further simplifying the approximate test.

6 Related Work

There exist many approaches to WSC (e.g., [12, 13]) that compile the composition problem into a planning formalism. These approaches also assume exact matches of input/output types, therefore ignoring the background ontologies. [2] focuses on information gathering at composition time. Two approaches explore how to adapt formalisms from hand-tailored planning, namely Golog [10], respectively HTN planning [16], for WSC. Both approaches are capable of composing services involving control constructs (loops, branches). There is one fully automatic approach handling control constructs [11]. There, BDD-based search techniques are exploited to obtain complex solutions; input/output type matches are assumed to be exact.

The requirements on matches are relaxed in the following works. In [1], techniques are introduced that disambiguate concept names during WSC. [15] extends the classical HTN (Hierarchical Task Network) planning to work with OWL-S processes, and therefore performs matching with respect to OWL-DL ontologies. However, the type of matches considered is plug-in, i.e. a task will match only if all its preconditions are fulfilled. This approach further differs from ours in that, similar to [10, 16], it uses workflow

$\mathcal{W}$	lookahead = 1		lookahead = 2		lookahead = 3		lookahead = 4	
	T	GS	T	GS	T	GS	T	GS
6	5.968	54	2.406	31	1.641	20	1.641	20
6+5	459.563	1082	62.020	295	3.422	26	3.422	26
6+10	>30 min		110.078	483	4.750	41	4.766	41
6+15	>30 min		>30 min		171.390	458	164.439	123
6+20	>30 min		>30 min		134.829	362	95.000	190
6+25	>30 min		>30 min		221.703	530	157.860	211
6+30	>30 min		>30 min		>30 min		>30 min	

Table 2: Results when varying the lookahead

templates, describing the outline of activities that need to be performed, to achieve composition.

Partial matches are addressed in [3], a fully automatic approach to WSC. The approach uses numeric intervals to encode a sub-concept hierarchy, and performs matching (as well as service discovery) based on those intervals, by asking whether one interval (a service output) is fully contained in the union of a set of intervals (service inputs). Both forward and backward search are performed (however, for backward search only complete matches are considered). There are several differences to our approach. First, in [3] the functionality of web services is described only with typed inputs and outputs, without preconditions and effects. The main consequence is that it is not possible to encode the relation of outputs with respect to inputs, feature present in the forward effects. Second, our ontologies are more general, taking arbitrary clausal form instead of a sub-concept hierarchy. Third, we guide the search using a heuristic function, instead of performing a blind depth-first search.

7 Conclusions and Future Work

We presented a tool able to solve $\mathcal{WSC}|_{fwd}$ tasks. We defined the search space of the tool and optimized the basic procedures. Further, we designed algorithms for guiding the search. We compute the heuristic value of forward effects beliefs as an approximation of the number of steps necessary to reach the goal. With a backchaining step, we identify a relaxed solution, from which we extract the most promising applications.

Our future work includes adding a parallelization step, as well as designing and evaluating less costly heuristic functions. In the long term, we will incrementally enrich the language our tool accepts, accordingly adapting the algorithms. For a start, some generalizations of $\mathcal{WSC}|_{fwd}$ are possible without losing Proposition 3.1. Most importantly, instead of requiring that *every* effect literal involves a new constant, one can postulate this only for literals that may actually be affected by the ontology.

References

[1] R. Akkiraju, B. Srivastava, I. Anca-Andreea, R. Goodwin, and T. Syeda-Mahmood. Semaplan: Combining planning with semantic matching to achieve web service composition. In *Proc. ICWS'06*.

[2] T.-C. Au and D. Nau. The incompleteness of planning with volatile external information. In *Proc. ECAI'06*.

[3] I. Constantinescu, B. Faltings, and W. Binder. Large scale, type-compatible service composition. In *ICWS*, 2004.

[4] D. Martin et al. OWL-S: Semantic markup for web services. In *SWSWPC*, 2004.

[5] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning, II: The DLVK system. *AI*, 144(1-2):157–211, 2003.

[6] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer-Verlag, 2006.

[7] J. Hoffmann, P. Bertoli, and M. Pistore. Web service composition as planning, revisited: In between background theories and initial state uncertainty. In *AAAI*, 2007.

[8] J. Hoffmann and R. Brafman. Conformant planning via heuristic forward search: A new approach. *AI*, 170(6–7):507–541, 2006.

[9] J. Hoffmann, I. Weber, J. Scicluna, T. Kaczmarek, and A. Ankolekar. Combining scalability and expressivity in the automatic composition of semantic web services. Accepted for ICWE'08.

[10] S. McIlraith and T. C. Son. Adapting Golog for composition of semantic Web services. In *Proc. KR-02*.

[11] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. In *Proc. IJCAI'05*, 2005.

[12] S. Ponnekanti and A. Fox. SWORD: A developer toolkit for web services composition. In *WWW*, 2002.

[13] M. Sheshagiri, M. desJardins, and T. Finin. A Planner for Composing Services Described in DAML-S. In *Proc. AAMAS'03*, 2003.

[14] A. Sirbu and J. Hoffmann. Towards scalable web service composition with partial matches. Technical report, University of Innsbruck, 2008. Available from http://www.sti-innsbruck.at/results/publications/technical-reports/.

[15] E. Sirin, B. Parsia, and J. Hendler. Template-based composition of semantic web services. In *AAAI Fall Symposium on Agents and Search*, 2006.

[16] 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.

[17] M. Winslett. Reasoning about actions using a possible models approach. In *AAAI*, 1988.