Polynomial-Time Reasoning for Semantic Web Service Composition

Jörg Hoffmann and James Scicluna
DERI Innsbruck
Innsbruck, Austria
⟨first⟩.⟨last⟩@deri.org

Tomasz Kaczmarek
Poznan University of Economics
Poznan, Poland
t.kaczmarek@kie.ae.poznan.pl

Ingo Weber
SAP Research
Karlsruhe, Germany
ingo.weber@sap.com

Abstract

Automatic composition of semantic web services should make use of the ontology in which the services are specified. While the approaches can strongly benefit from doing so, they have to deal with the frame and ramification problems, necessitating worst-case exponential reasoning even to determine the outcome of applying a single web service. The existing approaches to composition either ignore the background ontology, matching web services based on concept names and hence removing the need for reasoning; or they employ full-scale reasoning and suffer from the unavoidable performance deficiencies. In our work, we instead look for interesting classes of ontologies where the required reasoning is polynomial. We define a formalism for semantic web service composition. We present polynomial-time methods for dealing with several of the most commonly used ontology modelling constructs; further extensions are possible. We prove that our methods are correct. We are currently developing an implementation of our techniques.

1 Introduction

Automatic web service composition (WSC), based on semantic descriptions, is one of the hot topics in the Web Services and Semantic Web areas. Semantic web services are pieces of software advertised with a formal description of what they do; composing them means to link them together in a way satisfying a complex user requirement. WSC is widely recognized for its huge economic potential, in areas such as enterprise application integration. Two initiatives for semantically describing web services are OWL-S (e.g. [1, 4]) and WSMO (e.g. [7]). In the OWL-S “service profile”, and the WSMO “capability”, services are described akin to AI planning formalisms, specifying input/output parameters, preconditions, and effects. We focus on this kind of semantic descriptions in our work.

One main obstacle in WSC is that the web services must be composed in the context of the background ontology in which they are specified. This brings about what AI refers to as the frame and ramification problems. The latter necessitate worst-case exponential reasoning even to determine the outcome of applying a single web service. Consequently, the existing approaches to WSC either ignore the background ontology, matching web services based on concept names and hence removing the need for reasoning (e.g. [16, 14, 15, 2, 3]); or they employ full-scale general reasoners and hence suffer from the unavoidable performance deficiencies (e.g. [8, 17, 13]).

In our work, we explore a third direction, which was given little attention as yet. We look for interesting classes of ontologies where the reasoning required for the composition is polynomial. We define a formalism for semantic web service composition. We present polynomial-time methods for dealing with several of the most commonly used ontology modelling constructs; further extensions are possible. We prove that our methods are correct. We are currently developing an implementation of our techniques.

1 The motivation for this arrangement are results from AI Planning, indicating that forward search with parallel operators is inefficient.
of web services. Discovery is performed as a pre-process that identifies the set of available services from which the solution should be composed. Note that this arrangement is needed for defining a heuristic function: the heuristic estimation must be based on the available services, and cannot be specified sensibly if that set is not known.

The remainder of the paper is structured as follows: Section 2 introduces our formalism for WSC. Section 3 explains how forward search proceeds, and Section 4 introduces our polynomial time algorithms for computing state transitions. Section 5 discusses the most closely related work, and Section 6 concludes.

2 Composition Formalism

Our composition formalism is denoted with WSC; it follows the relevant literature from the AI actions and change research, e.g., [20, 9]. We consider plug-in matches. With this form of matching, a web service is applicable if it matches all possible situations. The following is an illustrative example.

Example 1 Our goal is to compose a web service that accepts as input a constant of type ‘holidayPlan’ and returns another constant of type ‘flightTicket’. The background ontology defines the concepts ‘holidayPlan’, ‘holidayTrip’, ‘flight’ and ‘flightTicket’ and states that ‘holidayTrip’ is a sub-concept of ‘flight’ (i.e., holidayTrip ⊆ flight). A web service ‘MakeTrip’ is available that transforms ‘holidayPlan’ into ‘holidayTrip’. Another web service ‘BookFlight’ transforms ‘flight’ into ‘flightTicket’. The match of the input of ‘BookFlight’ against the output of ‘MakeTrip’ is plug-in: the class of accepted inputs of ‘BookFlight’ subsumes the class of outputs of ‘MakeTrip’ but it is not identical to it.

Though the notation of our formalism is borrowed from AI Planning, we emphasize that we do not compile WSC into a planning problem like some previous works (e.g. [16, 2, 18]) did. Rather, our formalism is tailored to naturally correspond to WSC, and we develop targeted technology. In the formalism, web services correspond to planning “operators”. Their input/output behavior maps to input/output parameters, on which preconditions and effects are specified. This closely matches to the specification of web services at the OWL-S service profile level and to the WSMO capability level. The domain ontology turns into a “background theory”. The goal capability’s precondition turns into sets of “initial literals” and “initial constants”; the goal capability’s effect turns into a “goal condition”.

The background theory incurs the frame and ramification problems when performing state transitions. We deal with those problems via a standard notion of possible worlds with minimal changes, as first introduced in [20].

Let us give the formal notation. We assume a supply of logical predicates, a supply of variable names, and an infinite supply of constant names; we will denote predicates with $G$, $H$, $I$, variables with $x, y$, and constants with $c, d, e$. Literals are possibly negated predicates whose arguments are variables or constants; if all arguments are constants, the literal is ground. Given a set $X$ of variables, we denote by $L^X$ the set of all literals which use only variables from $X$. If $I$ is a literal, we write $l[I]$ to indicate that $I$ has the variable arguments $X$. If $X = \{x_1, \ldots, x_k\}$ and $C = \{c_1, \ldots, c_k\}$, then by $l[c_1, \ldots, c_k/x_1, \ldots, x_k]$ we denote the respective substitution, abbreviated as $l[C]$. In the same way, we use the substitution notation for any construct involving variables. Slightly abusing notation, we identify a vector with the set of constants appearing in it. We refer to positive ground literals as propositions, denoted with $p, q$.

To be able to naturally model updates on attribute values in web service descriptions, we introduce a special treatment for those. Namely, if $G$ is a binary predicate (a predicate of arity 2), and $x$ is a variable, then $G(x)$ is an attribute selector. Attribute selectors are allowed as arguments of literals. Note that, by this definition, attribute selectors can not be nested. If $X_1$ and $X_2$ are sets of variables, then we denote by $L^{X_1 \cup p(X_2)}$ the set of all literals whose arguments are either constants, or variables from $X_1$, or attribute selectors whose arguments are from $X_2$; by $L^{X_1}$ we denote the set of all literals whose arguments are either constants or variables from $X_1$, i.e., if no $G(X_2)$ is specified then no attribute selectors are allowed. Attribute selectors will be allowed in operator (web service) descriptions, and their semantics will be defined relative to the situation in which the operator is applied. Intuitively, the attribute selectors serve as macros that are expanded with the image of the respective attribute in the current situation. Note that, hence, while syntactically the attribute selectors are somewhat reminiscent of function symbols, their meaning is quite different.

A theory is a conjunction of closed (no free variables) first-order formulas, where the literals do not contain any attribute selectors. A clause is a disjunction of literals with universal quantification on the outside, e.g. $\forall x.(-G(x) \lor H(x) \lor I(x))$. A theory is clausal if all its conjuncts are clauses. A clause is a subsumption relation if it has the form $\forall x.(-G(x) \lor H(x))$, i.e., all literals have arity 1 and share the same variable, and every clause contains exactly one positive and exactly one negative literal. A clause is a cardinality restriction if it has the form $\forall x, y_1, \ldots, y_k.(-G(x, y_1) \lor \ldots \lor -G(x, y_{k+1}) \lor y_1 = y_2 \lor y_1 = y_3 \lor \ldots \lor y_k = y_{k+1})$; such a clause demands that, for any given $x$, there are at most $k$ different constants $y$ that are in relation $G(x, y)$. This is a common restriction in ontologies, where the cardinality—the number of values in the image—of attributes is often bounded (to 1, e.g.). To simplify notation, we will refer to a constraint of the above
form as \( \text{image}(G) \leq k \). A theory is a subsumption hierarchy with cardinality restrictions if all its conjuncts are either subsumption relations or cardinality restrictions.

An operator \( o \) is a tuple \((X_o, \text{pre}_o, Y_o, \text{eff}_o)\), where \( X_o \), \( Y_o \) are sets of variables, \( \text{pre}_o \) is a conjunction of literals from \( \mathcal{L}^{X_o, p(X_o)} \), and \( \text{eff}_o \) is a conjunction of literals from \( \mathcal{L}^{X_o, Y_o, p(X_o)} \). The intended meaning is that \( X_o \) are the inputs and \( Y_o \) the outputs, i.e., the new constants created by the operator; \( \text{pre}_o \) is the precondition, \( \text{eff}_o \) the effect (also sometimes referred to as the postcondition in the literature); attribute selectors can be used to refer to the images of the attributes of the inputs. Before an operator can be applied, it must be transformed into an action. An action \( a \) is a pair \((\text{pre}_a, \text{eff}_a)\) of conjunctions of ground literals. Namely, for an operator \( o \) and tuples of constants \( C_o \) and \( E_o \), an action \( a \) is given by \((\text{pre}_a, \text{eff}_a) = (\text{pre}_o, \text{eff}_o)[C_o/X_o, E_o/Y_o]\). That is, the operator’s inputs are instantiated with the tuple of constants \( C_o \), and the operator’s outputs are instantiated with the tuple of constants \( E_o \), where we require that the constants in \( E_o \) are pairwise different – it makes no sense to output the same new constant “twice”. Before an action can be applied, its attribute selectors must be expanded. This will be defined formally further below.

We can now define tasks. These are tuples \((P, T, O, C_0, \phi_0, \phi_G)\). Here, \( P \) is a set of predicates; \( T \) is the background theory; \( O \) is a set of operators; \( C_0 \) is a set of constants, the initial constants supply; \( \phi_0 \) is a conjunction of ground literals, describing the possible initial worlds; \( \phi_G \) is a conjunction of literals describing the goal worlds, e.g. \( G(x) \land H(y) \), where the variables will be interpreted as existentially quantified on the outside.

2All predicates are taken from \( P \), and all constants are taken from \( C_0 \). All constructs (e.g. sets and conjunctions) are assumed to be finite.

In what follows, assume we are given a task \((P, T, O, C_0, \phi_0, \phi_G)\). The semantics of these tasks – the definition of what a solution is – relies on a notion of beliefs, where each belief is a set of worlds that are considered possible. We first formally define the latter. A world \( w \) is a pair \((C_w, I_w)\) where \( C_w \) is a set of constants, and \( I_w \) is a \( C_w \)-interpretation, i.e., an interpretation of the predicates \( P \) over the constants \( C_w \). Here, \( C_w \) contains the constants that exist in \( w \); this serves to keep track of the creation of new constants. \( I_w \) is a truth value assignment to all propositions formed from \( P \) and \( C_w \). We write \( w \models \phi \) for \( I_w \models \phi \), where the quantifiers in \( \phi \) are restricted to \( C_w \).

As stated, beliefs are sets of possible worlds; i.e., at each point in time, our uncertainty about the true state of the world is expressed in terms of the set of worlds that may be possible. The initial belief \( b_0 \) is undefined if \( T \land \phi_0 \) is not satisfiable; else, \( b_0 := \{ w \mid C_w = C_0, w \models T \land \phi_w \} \). An solved belief is a belief \( b \) s.t. ex. a tuple \( C \) of constants s.t., for all \( w \in b, w \models \phi_G[C] \). It remains to define how the application of actions affects beliefs. This is a rather involved definition, relying on three more basic notions: expansion of attribute selectors, parallel actions, and transitions over worlds. We will formally define these notions, in that order, below. For the sake of readability, it is convenient at this point to explain how these notions will be used to define belief transitions. Given a belief \( b \) and an action \( a \), the expansion of \( a \) in \( b \) is an action without attribute selectors, denoted \( \text{Expand}(b, a) \). Parallel actions are sets of non-conflicting actions, denoted \( A \), all applied at the same time; we use the notation \( \text{Expand}(b, A) \) with the obvious meaning. Given a world \( w \) and an expanded parallel action \( A \), the result of applying \( A \) in \( w \) is a set of worlds, denoted \( \text{res}(w, A) \). Using these notions, we define belief transitions. Assume a belief \( b \) and a parallel action \( A \); the result of applying \( A \) in \( b \) is denoted with \( \text{res}(b, A) \). This is undefined if there exists a world \( w \in b \) so that \( \text{res}(w, \text{Expand}(b, A)) \) is undefined, or so that \( \text{res}(w, \text{Expand}(b, A)) = \emptyset \). Else, \( \text{res}(b, A) := \bigcup_{w \in b} \text{res}(w, \text{Expand}(b, A)) \).

The res function is extended to sequences of parallel actions in the obvious way. A solution is a sequence \( \{A_1, \ldots, A_n\} \) s.t. \( \text{res}(b_0, \{A_1, \ldots, A_n\}) \) is a solved belief. Note here that plug-in matches correspond to the requirement that \( \text{res}(b, A) \) is defined (and non-empty) across all \( w \in b \); this is the formal equivalent to the intuitive statement that “A must be applicable in all possible situations”. To obtain a notion of partial matches, one could simply weaken the requirement to state that \( A \) must be defined in at least one of the \( w \in b \).

We subsequently fill in the more basic definitions. First, we formally define how attribute selectors in actions are expanded. Say \( a \) is an action and \( b \) is a belief. Then the expansion of \( a \) in \( b \) is \((\text{Expand}(b, \text{pre}_a), \text{Expand}(b, \text{eff}_a))\), where the \( \text{Expand} \) function is defined as follows. For a belief \( b \) and a conjunction \( L \) of ground literals, \( \text{Expand}(b, L) = L \) if no attribute selectors appear in \( L \). Else, say \( L = l_1 \land \ldots \land l_k \). Then \( \text{Expand}(b, L) = \bigwedge_{e \in b \land l_e \land l_1} (G(e) \land l_{k+1} \land \ldots \land l_n) \). In words, the attribute selector \( G(e) \) is replaced with all constants \( c \) that satisfy \( G(e), c \) in the entire belief, where every \( c \) yields a new literal. This is repeated until no more attribute selectors remain.

Parallel actions are sets of actions which are applied at the same point in time. We require that, in a parallel action \( A \), each pair of actions is non-conflicting: a pair \( a, a' \) of actions is non-conflicting if \( E_a \cap E_{a'} = \emptyset \) – the actions do not try to generate the same constants – and \( \text{eff}_a \land \text{eff}_{a'} \), \( \text{pre}_a \land \text{pre}_{a'} \), \( E_a \land E_{a'} \), and \( \text{eff}_a \land \text{pre}_{a'} \) are all satisfiable – the effects.

2The former happens if the expanded \( A \) is not applicable to \( w \), the latter happens in case of unresolvable conflicts between \( A \)'s expanded effects and the background theory; more below.
and preconditions of the actions are not in direct conflict.

We next define the outcome of applying parallel actions in worlds, \( res(w, A) \). Note that the definition ignores attribute selectors, since those have already been expanded when calling \( res(w, A) \). Given a world \( w \) and a parallel action \( A \), \( A \) is applicable in \( w \) if, for every \( a \in A \): \( w \models \text{pre}_a \), \( C_a \subseteq C_w \), and \( E_a \cap C_w = \emptyset \). That is, we require for every \( a \in A \) that the precondition is satisfied, that the inputs exist, and that the outputs do not yet exist. If \( A \) is not applicable in \( w \), then \( res(w, A) \) is undefined. Otherwise, we set

\[
    res(w, A) := \{ (C', I') \mid C' = C_w \cup \bigcup_{a \in A} E_a, \quad I' \in \text{min}(w, C', T \land \bigwedge_{a \in A} \text{eff}_a) \}
\]

Here, \( \text{min}(w, C', \phi) \) is the set of all \( C' \)-interpretations that satisfy \( \phi \) and that are minimal with respect to the partial order defined by \( I_1 \leq I_2 \) if for all propositions \( p \) over \( C_w \), if \( I_2(p) = I_2(p) \) then \( I_1(p) = I_1(p) \). In words, a \( C' \)-interpretation is in \( \text{min}(w, C', \phi) \) iff it satisfies \( \phi \), and as is close to \( w \) as possible. Note that \( \text{min}(w, C', \phi) \) is empty in case \( \phi \) is unsatisfiable – i.e., if \( \bigwedge_{a \in A} \text{eff}_a \) is in conflict with \( T \) then the outcome of \( A \) is empty.

Example 2 (Plug-in Matches, Formalized) Re-consider Example 1. It is formulated in WSC as follows.

- \( \mathcal{P} = \{ \text{holidayPlan}, \text{holidayTrip}, \text{flight}, \text{flightTicket} \} \)
  \( T = \forall x. (\neg \text{holidayTrip}(x) \lor \text{flight}(x)) \)

- \( \mathcal{O} \text{ consists of } \text{MakeTrip} = ((x), \text{holidayPlan}(x), \{ y \}, \text{holidayTrip}(y)) \) and \( \text{BookFlight} = ((x), \text{flight}(x), \{ y \}, \text{flightTicket}(y)) \)

- \( C_0 = \{ \text{myPlan} \}; \phi_0 = \text{holidayPlan(myPlan)}; \phi_G = \text{flightTicket}(x) \)

The initial belief \( b_0 \) consists of all worlds \((C, I)\) where \( C = \{ \text{myPlan} \} \), and \( I(\text{holidayPlan(myPlan)}) = T \) (otherwise, \( I \) is unconstrained). If we apply \( \text{MakeTrip} \) to \( b_0 \), instantiated with \( \text{myPlan} \) as the input and a new constant \( \text{someTrip} \) as the output, then the result that we get is the belief \( b_1 \) containing all worlds \((C', I')\) where \( C' = \{ \text{myPlan}, \text{someTrip} \}, I'(\text{holidayTrip}(\text{myPlan})) = T, I'(\text{holidayTrip}(\text{someTrip})) = T, \) and \( I'(\text{flight}(\text{someTrip})) = T \). Note that the last fact is implied by the theory. When applying \( \text{BookFlight} \) to this, we obtain a solution: for every \( w \in b_1 \), the precondition of \( \text{BookFlight} \) is satisfied.

3 Forward Search

The main loop of our search for a solution is a forward search. The algorithm is straightforward; we include it here to show how the reasoning algorithms, specified in the next section, are put to use. Pseudo code is in Figure 1.

\( s_0 := \text{get-start-state}() \)

if \( s_0 \) is undefined then
  return “initial literals are inconsistent”
endif

open-list := \( s_0 \)
while TRUE do
  if open-list is empty then
    return “no plan exists”
  endif
  \( s := \text{remove-front(open-list)} \)
  if \( \text{is-solution}(s) \) then return path leading to \( s \) endif
  \( \text{appl} := \text{find-all-actions-with-satisfied-precondition}(s) \)
  for all \( a \in \text{appl} \) do
    \( s' := \text{get-result-state}(s, a) \)
    if \( s' \) is undefined then continue endif
    \( h := \text{heuristic-function}(s') \)
    insert-queue(open-list,queueing-function,\( s', h \))
  endfor
endo"
Two functionalities need to be implemented: (1) A function get-start-state that computes the start state \( s_0 \) to initialize the search. (2) A function get-result-state that, given a state \( s \) and a parallel action \( A \), computes the result state, i.e., the outcome of applying \( A \) in \( s \). We specify polynomial-time algorithms for both. Pseudo code for (1) is given in Figure 2. Pseudo code for (2) is given in Figure 3. We next explain the pseudo-code, then we prove correctness.

**Procedure get-start-state()**

1. \( L := \{ l \mid l \text{ appears in } \phi_0 \} \)
2. for all \( \text{image}(G) \leq k \text{ in } T \) do
3. if \( \exists \text{ c.s.t. } \{d \mid G(c,d) \in L\} > k \) then
4. return undefined
5. endfor
6. endwhile
7. while \( \text{TRUE} \) do
8. \( L' := L \)
9. \( L' := L' \cup \{H(c) \mid \forall x. (\neg G(x) \lor H(x)) \in T, G(c) \in L\} \)
10. \( L' := L' \cup \{-G(c) \mid \forall x. (\neg G(x) \lor H(x)) \in T, -H(c) \in L\} \)
11. if \( L = L' \) then break else \( L := L' \) endif
12. endwhile
13. if \( \exists \text{ l.s.t. } l \in L \text{ and } \bar{l} \in L \) then
14. return undefined
15. endif
16. return \( (C_0, L) \)

**Figure 2. Computing the start state.**

Consider first Figure 2. Lines (2) to (6) check if any cardinality constraint is violated by the initial literals. If so, then the initial state is contradictory and an undefined state is returned. Otherwise, the procedure performs a simple propagation operation over the subsumption relations in the theory, incrementally increasing a set of literals \( L \). Line (9) does upwards propagations stating that a constant belonging to a sub-type also belongs to a super-type; line (10) does the corresponding downwards propagations. The propagations are repeated until a fix-point is reached (line 7); trivially, this will be the case after a polynomial number of iterations. The initial set \( L \) is formed by the literals in the initial conjunction \( \phi_0 \) of the task (line 1). After the propagation fix-point, \( L \) also contains all literals implied from this by the background theory. If this set of literals is contradictory, an undefined state is returned: with \( \bar{l} \) for a literal \( l \), we denote the inverse literal. If there is no contradiction, \( L \) is returned, together with the initial constants set \( C_0 \) of the panning task.

Consider now Figure 3. Line (1) initializes the set \( L \) with the expanded action effects. Here, \( \text{Expand} \) selects the attribute relations from \( L_s \) (rather than from the intersection of the worlds in the belief as defined before). Note that the computation of \( \text{Expand} \) is worst-case exponential in the arity of the involved predicates, since one can put several attribute selectors into a single literal. However, i) the latter seems rather artificial so the worst-case is probably pathological; and ii) it is safe to assume that a maximum predicate arity is fixed. In particular, in Description Logics, the maximum predicate arity is 2.

Similar to the previous function, the algorithm next checks whether the (expanded) action effects already violate any cardinality constraint. If that is not the case, the procedure performs a propagation fix-point as before, this time initialized with the conjunction of all effects of the parallel action. The set \( L \) accumulates, through the propagation, all literals implied from this by the background theory. Line (13) then uses this set to test whether the action contradicts the background theory; line (14) tests whether one action in \( A \) is not applicable; if either test succeeds, the function terminates stating that the result is undefined.
Line (17) is the first of two steps taking the previous state $s$ into account. $L$ is extended with all literals from $L_s$, except those that are contradicted by $L$. We emphasize here that, while this computation seems like a straightforward thing to do, it is actually quite remarkable that this works in our context. Recall that what we need to do is, based on the literals $L_s$ true in the previous belief, $b_s$, compute the literals $L$ true in the successor belief, $b$. This is a complex computation in general, c.f. Section 2. Interestingly, given our restrictions on the background theory, we can simply keep all literals $L_s$ except for those for which we explicitly derived a contradiction; see Theorem 1. Importantly, the theorem does not hold in general. E.g., we constructed a counter example using a single clause with 3 literals.

A final part of the algorithm, lines (18) to (25), performs some extra steps necessary to deal with cardinality restrictions. Namely, it may be that, together with the effects, the properties inherited from the previous state violate a cardinality constraint. In that case, all respective literals inherited from the previous state are removed. This is adequate because, in this situation, it will not be known in the successor belief which previous attributes still hold. On the other hand, if the action effects assigned all possible attributes — if the number of attributes is equal to the maximum number specified in the cardinality restriction — then we know that none of the previous attributes persists, hence we know that they are all false, hence line (22) inserts the respective negated literals. Finally, the result state is returned as the literals $L$ together with the resulting constants, which are simply computed as the constants $C_s$ from $s$ plus the constants created by the parallel action.

The following theorem states that our algorithms are correct, i.e., that the computed states $s$ represent the corresponding beliefs as required. The proof is given in Appendix A. For a belief $b$, we denote $\text{Lits}(b) := \{l \mid l \text{ literal over } \mathcal{P} \text{ and } C_w, I_w \models l\}$. Theorem 1 Assume a WSC task $(\mathcal{P}, \mathcal{T}, \mathcal{O}, C_0, \phi_0, \phi_G)$ where $\mathcal{T}$ is a subsumption hierarchy. Assume a sequence of parallel actions $(A_1, \ldots, A_n)$. Say $b$ is the belief state after executing $(A_1, \ldots, A_n)$. Say $s$ is the state after executing $(A_1, \ldots, A_n)$ using the algorithms from Figures 2 and 3. Then:

- (1) $b$ is defined iff $s$ is defined;
- (2) If $s$ is defined, then for all $w \in b$: $C_w = C_s$;
- (3) If $s$ is defined, then $L_s = \text{Lits}(b)$.

Next, we show that solutions in the new search space correspond to solutions in the original one. We define a solution state as a state $s$ where there exists a tuple $C$ of constants in $C_s$ for which $\phi_G[C/X] \subseteq L_s$. That is, in a solution state we can instantiate the goal variables in a way so that all goal literals are known to be true.

Corollary 1 Assume a WSC task $(\mathcal{P}, \mathcal{T}, \mathcal{O}, C_0, \phi_0, \phi_G)$. Assume a sequence of parallel actions $(A_1, \ldots, A_n)$. Say $s$ is the state after executing $(A_1, \ldots, A_n)$ using the function from Figures 2 and 3. If $s$ is a solution state, then $(A_1, \ldots, A_n)$ is a plan.

Proof: By definition, $(A_1, \ldots, A_n)$ is a plan iff for all $w \in \text{res}(b_0, (A_1, \ldots, A_n)) : w \models \phi_G$. Here, $\phi_G$ is an existentially quantified conjunction of literals. By definition, if $s$ is a solution state, we can find an instantiation $C$ of all variables so that $\phi_G[C/X] \subseteq L_s$. With Theorem 1, this proves the claim.

Our next step will be to further extend the set of allowed constructs in the background theory. In particular, we have already started exploring the following three extensions:

- **Lower bound cardinality restrictions.** So far, cardinality restrictions give only upper bounds on the number of allowed attribute values. A lower bound cardinality restriction is given by a formula of the form $\forall x. \exists y_1, \ldots, y_k. (G(x, y_1) \land \ldots \land G(x, y_k) \land y_1 \neq y_2 \land y_1 \neq y_3 \land \ldots \land y_1 \neq y_k)$. Such formulas can be handled in a way very similarly to what we have described above for upper bound cardinality restrictions. If not enough attribute values are set, then the respective belief contains one world for every possibility to assign enough values. This can be covered with a set of case distinctions similarly as for upper bounds.

- **Attribute domain and image restrictions.** Another frequently used modeling construct are restrictions on the types (concepts) of instances that can take part in attribute relations; the domain regards the first argument of the attribute, the image the second one. More formally, a domain restriction takes the form of a clause $\forall x. y. (\neg p(x, y) \lor q(x))$, and an image restriction takes the form of a clause $\forall x. y. (\neg p(x, y) \lor q(y))$. For example, a domain restriction is given every time an ontology declares an attribute as part of a concept. The logical consequences of domain and image restrictions can be obtained by simple propagation methods, which can be naturally integrated into our methods dealing with subsumption hierarchies.

- **2-clauses.** More generally, one can deal with any clause that contains at most 2 literals. Namely, in subsumption hierarchies, every logical consequence of a set of clauses can be derived by polynomial-time reasoning (unit propagation). This is the key property that we exploit for our polynomial-time algorithm. Now, the same property holds for any propositional clausal theory, as long as each clause contains at most 2 literals. Our algorithms can be extended by integrating the appropriate reasoning techniques.
5 Related Work

In [3], WSC is reduced to satisfiability in Propositional Dynamic Logic (which is EXPTIME complete) and hence the “reasoning” performs the entire composition rather than only the search state updates, as in our case. A closer relative to our work is the investigation of instance level updates in DL-Lite [10], which like our work formalizes a notion of a background theory – a description logic (DL) “TBox”, in that case – and investigates what happens if instance data is updated – at the DL “ABox” level. The allowed instance data updates are quite similar to ours, and the focus is on identifying tractable classes. ⁴ The main differences to our work lie in the formalization, and in the considered background theories. [10] assume that each “search state” – DL ABox – is a complete characterization of the respective “belief”, in that the belief must be equal to the set of models of the ABox. This restricts their work to beliefs that are actually representable in that way. By contrast, in our formalism the search states capture only the intersection of a belief, not its entirety, and hence we do not have that problem. Regarding the considered background theories, [10] allow subsumption relations, plus other DL constructs such as unqualified existential quantification (of basic concepts). However, cardinality restrictions and attribute selectors cannot be expressed; more generally, whereas we explore certain restricted kinds of disjunctions, [10] disallow them. Note that the different modelling constructs considered stem from the different work contexts. In difference to [10], our work is concerned with development of an actual WSC tool, and hence driven by the modelling constructs most typically used in describing web services.

Of the existing WSC tools, the closest relative to our work is “type-based service composition” [6, 5]. In this work, the background ontology is a subsumption hierarchy. This is compiled into intervals, where each interval represents a concept and the contents are arranged to correspond to the hierarchy; note the difference to our work, where we investigate more general theories. The intervals are used for the matching during composition. Composition is interleaved with discovery, and search proceeds in a depth-first fashion, with no heuristic information. Herein lies another difference to our ongoing tool development.

6 Conclusion

Our work is concerned with the development of WSC techniques for Business Process Management. To make such techniques useful in practice, they need to be fast. Here the underlying reasoning is a key issue, since it must be done frequently during the search for a suitable composition. We address this point by identifying tractable classes of ontologies, which is a topic that has not been given much attention as yet. Our initial results are promising, and further possible extensions have been identified.

References


⁴[12] are also concerned with DL ABox updates, but in fairly expressive DLs where the computation is exponential in several parameters.
A Proof to Theorem 1

By induction over $n$. Base case, $n = 0$. Here the claims regarding whether $b$ and $s$ are defined, and regarding the constants, follow immediately. Regarding the literals, we need to prove that $L$, as output by the algorithm in Figure 2, is the same as $\text{Lits}(b_0)$. First, all $l \in L$ clearly follow from $\phi_0$ and $T$ so they are true in all $w \in b$ and hence $L \subseteq \text{Lits}(b_0)$. In the other direction, say $l \notin L$. This means that $l$ can be both satisfied and not satisfied given $\phi_0$ and $T$, so it occurs positively and negatively in worlds $w \in b$ and hence $L \supseteq \text{Lits}(b_0)$.

Inductive case, $n \rightarrow n + 1$. Denote with $b$ the belief state after executing $(A_1, \ldots, A_n)$ using $\text{res}$; denote with $s$ the search state after executing $(A_1, \ldots, A_n)$ using the algorithms from Figures 2 and 3; denote $A := A_{n+1}$; denote $b' := \text{res}(b, A)$; denote by $s'$ the output of the algorithm from Figure 3 on $s$ and $A$. Finally, denote with $E_1$ the set $L$ as computed by the algorithm from Figure 3 prior to reaching line (17), i.e., $E_1$ contains $\bigwedge_{a \in A} \text{Expand}(s, \text{eff}_a)$ and the literals implied from this by the theory; denote with $E_2$ the set $L$ as computed by the algorithm from Figure 3 in line (17), i.e., $E_2 = E_1$ plus the non-contradicted literals from $s$.

Consider property (1). With induction assumption, we know it holds for $b$ and $s$. Now, $b'$ is undefined iff $A$ is contradictory with, or not applicable to, some world in $b$. On the other hand, $s'$ is undefined iff a contradiction can be derived from $T \land \bigwedge_{a \in A} \text{Expand}(s, \text{eff}_a)$, or $A$ is not applicable to $s$. It is easy to see that applicability is the same on both sides, per induction assumption on properties (2) and (3). Regarding contradictions, the crucial observation is that, under plug-in matches, the actions do not have a “conditional effects semantics”: if the action is applicable, all its effects occur, independently of the world $w$ to which it is applied. Therefore, $A$ is contradictory with one $w \in b$ iff it is contradictory with all $w \in b$ iff the conjunction of its effects is not satisfiable together with the background theory. Since $T$ is a subsumption hierarchy with cardinality restrictions, the latter is the case iff a literal and its negation can be derived from $T \land \bigwedge_{a \in A} \text{Expand}(s, \text{eff}_a)$. Note here that, by definition, the expansions underlying $\text{Expand}$ in $w \in b$, and $\text{Expand}$ in $s$, are the same.

Property (2) follows simply from the induction assumption on property (2) and the fact that $A$ is applicable to all $w \in b$.

Finally, consider property (3). We need to prove that $L_{n+1} = \text{Lits}(b')$. We do so in two separate steps, one for arity 1 predicates – which may be affected by subsumption relations – and one for arity 2 predicates – which may be affected by cardinality restrictions; for predicates with other arities, the claim follows trivially. We denote the arity $k$ subset of a set $L$ of literals with $L^k$. Recall that $L_{n+1}^k = (E_1 \cup \{l \mid l \in L_s, l \notin E_1\})^{k+1}$. We need to prove that $L_{n+1}^1 = \text{Lits}(b')^1$. It is easy to see that $E_1^1 \subseteq \text{Lits}(b')^1$: all the effects and their implications must be true in every world of $b'$. On the other hand, a literal $l \in L_{n+1}^1$ is true in all worlds of $b$ by induction assumption on property (3). Further, if $l$ is not directly affected by $E_1^1$, then there is never a need to make $l$ false: due to the nature of subsumption hierarchies, and because action expansion is the same across all worlds in $b$, anything that may be affected in a single world is affected in all worlds, and hence appears in $E_1^1$. Hence $l$ will be also contained in every world of $b'$, and so we get $L_{n+1}^1 \subseteq \text{Lits}(b')^1$. For the other direction, say an arity 1 literal $l$ is true in every world of $b'$, i.e., $l \in \text{Lits}(b')^1$. This means that either $l$ is enforced by an effect/an effect implication, or it is carried over from $b$. In the former case, obviously $l \in E_1^1$. In the latter case, obviously $l \notin E_1^1$; with induction assumption on property (3), we have $l \in L_{n+1}^1$; hence, we have $l \in \{l \mid l \in L_{n+1}^1, l \notin E_1^1\}$. This concludes the argument.

We now prove that $L_{n+1}^2 = \text{Lits}(b')^2$. Say $\text{image}(G) \leq k$ is a cardinality restriction in $T$. If $\{d \mid G(c, d) \in E_2\} \leq k$, for all $c$, then the restriction is met and has no effect on the worlds in $b'$, Lines (18) to (25) of reasoning-resultstate cover this case correctly, namely by not doing anything. If, on the other hand, $\{d \mid G(c, d) \in E_2\} > k$, for a $c$, then the restriction is violated and the world transition function will remedy this by switching the values propositions had in the previous worlds. Namely, say $\{d \mid G(c, d) \in E_1\} = l_1$ (we have $l_1 \leq k$ by construction), and $\{d \mid G(c, d) \in E_2 \setminus E_1\} = l_2$ (we have $l_2 > 0$ by construction); i.e., the cardinality restriction is violated by $l_1 + l_2 - k$ constants $d$, for the constant $c$. We will distinguish two cases, $l_1 < k$ and $l_1 = k$. First, in any case, $b'$ will contain one world for every subset of $\{G(c, d) \mid G(c, d) \in E_2 \setminus E_1\}$ that has size $l_1 + l_2 - k$: removing such a subset is a minimal way to fix the violation of the restriction. Since every literal $G(c, d) \in E_2 \setminus E_1$ appears in one such subset, for every such literal there exists at least one $w \in b'$ so that $w \not\models G(c, d)$. Hence none of these literals is contained in $\text{Lits}(b')^2$. Now, if $l_1 < k$ then we have $l_1 + l_2 - k < l_2$, hence $l_2 = \{G(c, d) \mid G(c, d) \in E_2 \setminus E_1\} > l_1 + l_2 - k$ and the $l_1 + l_2 - k$ size subsets are strict subsets. Thus, for every literal $G(c, d) \in E_2 \setminus E_1$, there exists at least one $w \in b'$ so that $w \not\models G(c, d)$. This means that none of the literals $\neg G(c, d)$ is contained in $\text{Lits}(b')^2$. If, on the other hand, $l_1 = k$, then $l_1 + l_2 - k = l_2$, and the only $l_1 + l_2 - k$ size subset of $\{G(c, d) \mid G(c, d) \in E_2 \setminus E_1\}$ is the entire set. Thus, for every literal $G(c, d) \in E_2 \setminus E_1$, there does not exist a $w \in b'$ so that $w \models G(c, d)$. Hence, all of the literals $\neg G(c, d)$ are contained in $\text{Lits}(b')^2$. It is easy to see that lines (18) to (25) of reasoning-resultstate cover all these cases correctly. This concludes the argument.