On the aggregation problem for synthesized web services

Citation for published version:
Deng, T, Fan, W, Libkin, L & Wu, Y 2010, On the aggregation problem for synthesized web services. in *Database Theory - ICDT 2010, 13th International Conference, Lausanne, Switzerland, March 23-25, 2010, Proceedings.* ACM, pp. 242-251. https://doi.org/10.1145/1804669.1804700

Digital Object Identifier (DOI):
10.1145/1804669.1804700

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Early version, also known as pre-print

Published In:
Database Theory - ICDT 2010, 13th International Conference, Lausanne, Switzerland, March 23-25, 2010, Proceedings

On the aggregation problem for synthesized Web services

Ting Deng [a,*], Wenfei Fan [b,a], Leonid Libkin [b], Yinghui Wu [b]

[a] *State Key Laboratory of Software Development Environment, Beihang University, China*
[b] *University of Edinburgh, United Kingdom*

A R T I C L E   I N F O

Keywords:
Web services
Artifacts
Synthesis problem
Static analysis
Transducers

A B S T R A C T

The paper introduces and investigates the aggregation problem for synthesized mediators of Web services (SWMs). An SWM is a deterministic finite-state transducer defined in terms of templates for component services. Upon receiving an artifact, an SWM selects a set of available services from a library to realize its templates, and invokes those services to operate on the artifact, in parallel; it produces a numeric value as output (e.g., the total price of a package) by applying synthesis rules. Given an SWM, a library and an input artifact, *the aggregation problem* is to find a mapping from the component templates of the SWM to available services in the library that maximizes (or minimizes) the output. As opposed to the composition syntheses of Web services, the aggregation problem aims to optimize the realization of a given mediator, to best serve the users' need. We analyze this problem, and show that its complexity depends on the underlying graph of the mediator: while it is undecidable when such graphs contain even very simple cycles, it is solvable in single-exponential time in the size of the specification (i.e., the total size of the input SWM, library and artifact) for SWMs whose underlying graphs are acyclic. We prove several results of this kind, with matching lower bounds (NP and PSPACE), and analyze restrictions that lead to polynomial-time solutions. In addition, we study the aggregation problem for nondeterministic SWMs (NSWMs). We show that the aggregation problem for NSWMs with various underlying graphs retains the same complexity as its deterministic counterparts. We also provide complexity bounds for determining whether SWMs and NSWMs terminate.

1. Introduction

Fundamental research on Web services has mostly focused on service models, verification and composition. A variety of models have been proposed to specify the behaviors and interactions of Web services, based on finite-state automata [1–3], data-driven transducers [4–9] or recently, artifacts [10–13]. A number of verification problems have been studied to decide, e.g., whether a transaction with certain properties can be generated by a service, or whether two services are equivalent [2,4,6–9,11,14–16]. The composition synthesis aims to determine whether available services can be coordinated to deliver a requested service, by automatically generating a mediator. Complexity bounds on the composition problem have been established for various service models [1,3,5,8,17,18].

This paper studies a problem that has not yet received much attention, referred to as *the aggregation problem for Web services*. In practice a mediator is often predefined, in terms of templates for component services. Each template indicates a service of a certain functionality (e.g., for booking flight tickets or reserving hotel rooms), and is to be realized with an

* Corresponding author.
E-mail addresses: dengting@act.buaa.edu.cn (T. Deng), wenfei@inf.ed.ac.uk (W. Fan), Leonid@inf.ed.ac.uk (L. Libkin), Y.Wu-18@sms.ed.ac.uk (Y. Wu).

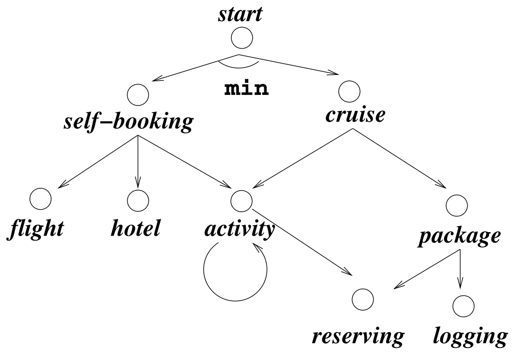

Fig. 1. An SWM specifying a travel planner.

available service. Provided that a mediator and a library of available services are *already in place*, a natural question concerns how to find an optimal realization of the mediator that best serves the users' need. That is, given user's input, we want to generate a composite service *on the fly* by selecting a set of available services from the library and realizing templates in the mediator with these services, such that certain values representing the user's interest are maximized (e.g., benefits) or minimized (e.g., price). We illustrate the problem by an example.

Example 1. Consider a mediator M_1 for planning a trip to Disney World. Users have two options, as shown in Fig. 1. (1) They may book a flight, reserve a hotel room, and arrange activities separately, all by themselves. (2) Alternatively, they may opt for a cruise package, with which the choices of hotels are limited. In either option, the users may repeatedly make reservations for activities, e.g., Disney World, scuba, to arrange a reasonable trip schedule.

The mediator M_1 is defined with component templates, e.g., *flight* and *activity*, which indicate services with a functionality for booking a flight and an activity, respectively. Such a template is to be realized with an available service having the required functionality. For example, *flight* can be realized with one of the online ticket booking systems launched by airlines or services such as Expedia and Priceline. Provided with travel dates, the available service that is chosen to realize *flight* returns the lowest airfare and reserves a ticket.

Provided travel dates and a list of free time slots, etc., the mediator is expected to explore both options to make a travel plan.

(1) For the first option, it ranges over available services for checking flights, hotel rooms and activities. It picks the ones that lead to the minimum cost C_1, which is the sum of the airfare, the accommodation cost and the costs of all the activities chosen.
(2) For the second option, it ranges over cruise packages, and for each package, it inspects its logging constraint and finds a hotel accordingly. It inspects activities as in the first option. The cost C_2 is the sum of the prices of the cruise, logging and activities.
(3) After both C_1 and C_2 are computed, the mediator returns the option with $\min(C_1, C_2)$. The option is reserved with the locked price [19], and recommended to the users. The users may then either decide to purchase the package, or cancel the reservation and repeat the process again. The actions are *not* committed *until* the users are ready to do so.

Observe that the templates in M_1 may be realized with possibly multiple available services. In this work we focus on how the mediator should realize its templates with the ones that lead to the lowest overall cost.

The aggregation analysis is not only of theoretical interest. The need for it is also evident in practice. In response to practical demand, there have been service providers looking into service selection based on the quality of services, e.g., the Océano project at IBM [20]. However, the issue has not yet received a formal treatment, from models for specifying aggregation syntheses to the complexity of the related problems.

The aggregation problem is, however, nontrivial. As illustrated in Example 1, there are typically multiple choices of available services to realize a component template. Furthermore, there is *data flow* [17] among the components, i.e., the output of a component is passed as *the input* to another; as a result, the realization of a component is dependent on the choice of the services for the components that invoke it. In addition, the control flow of the mediator may be complex, e.g., represented as a tree, DAG or a cyclic graph. These make this optimization problem rather challenging.

Contributions. We present a model to specify mediators with aggregation, formulate the aggregation problem, and establish complexity bounds on the problem for mediators of various structures, e.g., tree, DAG or cyclic graph.

Mediators with aggregation. We present a notion of *synthesized mediators for Web services* (SWMs), which extends mediators studied in [8] by incorporating aggregation synthesis. An SWM specifies a requested service that takes an artifact as input, and returns an aggregate value at the end. We consider artifacts that are updatable records representing the life-cycle of the processing of a requested service (see [10–12,21] for detailed discussions).

An SWM M is a deterministic finite-state transducer. Each state has a *transition rule* and a *synthesis rule*. The transition rule is specified with a precondition, component templates and successor states. Upon receiving an artifact, it checks whether the precondition is satisfied; if so it realizes the templates with available services in a library, invokes the services to operate on the artifact in parallel, and passes the updated artifacts downward to its successor states. The synthesis rule computes an aggregate value. It is defined in terms of a polytime-computable function on the aggregate values of the successor states, i.e., aggregate values are passed upward. The aggregate value generated in the start state of M is returned as (part of) the output of the service.

A formulation of the aggregation problem. An SWM M is realized with available services in a library L. A service in L is a function that takes an artifact as input and returns an (updated) artifact. A realization of M in L is a mapping ρ from the templates of M to L. Substituting service $\rho(\tau)$ for each template τ of M yields a *composite service* $M[\rho]$.

To ensure that composite services generated by a realization ρ are sensible, we consider *realization constraints* on ρ that specify what available services are allowed to realize a template.

Given an SWM M, an input artifact t, a library L, a realization constraint λ, the *aggregation problem*, denoted by $\mathsf{AGP}(M, L, \lambda, t)$, is to find a realization ρ of M in L that satisfies λ and maximizes (or minimizes) the output of $M[\rho]$ on the input artifact t.

Complexity bounds. The control flow of an SWM M can be depicted as a graph $G(M)$ of a form similar to Fig. 1, in which nodes are states of M and an edge (s_1, s_2) indicates that s_2 is a successor state of s_1. We establish lower and upper bounds on $\mathsf{AGP}(M, L, \lambda, t)$, all matching, for M of various structures. We show that $\mathsf{AGP}(M, L, \lambda, t)$ is undecidable when $G(M)$ is cyclic. In fact, for every cyclic graph G, the aggregation problem is undecidable over SWMs M so that $G(M) = G$. But when $G(M)$ is not cyclic (i.e., a DAG), the aggregation problem becomes decidable. Note that for many verification problems that ask questions about specifications (which are often expressed in temporal logics [22]), rather than data, single-exponential running time is viewed as acceptable (and in many cases unavoidable) [23]. We show that by forbidding cycles we get such acceptable algorithmic solutions: the problem is PSPACE-complete in the acyclic case, and the complexity drops further to NP-complete (but approximation-hard) when $G(M)$ is a tree.

In light of the intractability we also study special cases of $\mathsf{AGP}(M, L, \lambda, t)$. In particular, we give the complexity bounds for the problem when M is fixed but L varies, and when L is fixed while M may change. The former is to cope with a set of predefined mediators when the library L may take new services or drop obsolete services, and the latter is to accommodate the practical setting where a relatively stable library L serves various mediators. We show that the former simplifies the aggregation synthesis, e.g., $\mathsf{AGP}(M, L, \lambda, t)$ is in PTIME as opposed to PSPACE-complete for DAG-structured M. In contrast, the latter does not make our lives easier: the complexity bounds remain intact when L is fixed.

Aggregating nondeterministic mediators. In practice one often wants to use nondeterministic mediators. We introduce nondeterministic synthesis mediators (NSWMs) by extending SWMs, such that each state in an NSWM may have multiple pairs of transition and synthesis rules. That is, one is allowed to specify a variety of options for actions in a given situation. Upon receiving an artifact in a state, one of its transition rules is nondeterministically picked and applied if its precondition is satisfied, and its corresponding synthesis rule is used to compute the aggregate value.

We show that NSWMs do not make lives harder: $\mathsf{AGP}(M, L, \lambda, t)$ for NSWMs M has the same complexity as its deterministic counterparts. Specifically, it is undecidable, PSPACE-complete and NP-complete when $G(M)$ is cyclic, a DAG and a tree, respectively. Furthermore, the complexity results for the special cases of SWMs given above carry over to their nondeterministic counterparts.

Termination analysis. Finally, we investigate the termination problem for NSWMs. Given an NSWM M, an artifact t, a library L of available services and a realization constraint λ, *the termination problem* is to decide whether there exists a realization ρ of M in L such that ρ satisfies λ and a run of $M[\rho]$ terminates on the input t. We show that the problem is undecidable for SWMs M when $G(M)$ is cyclic, while a run of $M[\rho]$ always terminates for NSWMs M and valid realization ρ when $G(M)$ is acyclic.

To the best of our knowledge, this work is a first formal treatment of aggregation syntheses of Web services. Our results provide a comprehensive picture of complexity bounds for the aggregation problem, for deterministic and nondeterministic mediators. In addition, the proofs provide algorithmic insight for developing effective methods to conduct the syntheses. We summarize the main results of this work in Table 1.

Related work. This work extends [24] by including (1) the definition of nondeterministic mediators NSWMs, (2) the complexity bounds of the aggregation problem for various NSWMs, (3) the termination analysis for SWMs and NSWMs, and (4) the proofs of complexity results for the aggregation analysis of SWMs. Neither the results in Section 6 nor the proofs in Sections 4 and 5 were presented in [24].

Table 1
Complexity bounds on the aggregation problem $\mathsf{AGP}(M, L, \lambda, t)$.

Mediators M	$\mathsf{AGP}(M, L, \lambda, t)$ (the SWM case)	With fixed L	With fixed M	$\mathsf{AGP}(M, L, \lambda, t)$ (the NSWM case)
Tree-structured	NP-complete (Theorem 5) approximation-hard (Theorem 6)	NP-complete (Theorem 5) approximation-hard (Theorem 6)	PTIME (Proposition 8)	NP-complete approximation-hard (Theorem 11)
DAG-structured	PSPACE-complete (Theorem 7)	PSPACE-complete (Theorem 7)	PTIME (Proposition 8)	PSPACE-complete (Theorem 11)
Graph-structured	undecidable (Theorem 2)	undecidable (Theorem 2)	undecidable (Theorem 2)	undecidable (Corollary 10)

Several algorithms have been developed for selecting available services for service composition, based on the quality of services (QoS) [25–29]. Previous work on QoS differs from this work in the following aspects. (a) The criteria for QoS focus on system issues such as service response time, cost, reliability, availability, trust and bandwidth. In contrast, the aggregation problem is to maximize (or minimize) certain values in an artifact representing users' interest, which are the *data* processed by the services and are returned as output. (b) The complexity of the aggregation problem largely comes from data flow among component services, i.e., the output of one component is treated as the input of another. In contrast, data flow is not a major issue for previous work on QoS. (c) Previous work on QoS does not address how aggregation syntheses are expressed in Web services. Furthermore, previous results mostly consist of heuristic algorithms for estimating QoS and selecting available services accordingly; complexity bounds for service selection are not studied, except in [25]. A NP-complete bound was shown in [25] for optimal selection of available services, for pipelined (linear-structured) services based on a QoS model, in the presence of constraints on connecting a pair of services. The QoS model and the constraints of [25] are quite different from the aggregation syntheses and realization constraints studied in this work. Indeed, in the QoS settings the optimal selection problem remains in NP even for DAG-structured services [27], as opposed to the PSPACE-complete bound of this work.

Related to our work are also [30,31]: [30] proposes an approximation algorithm to find top-k flows of business processes *w.r.t.* an aggregate value, and [31] aims to find top-k execution flows with high likelihood in a probabilistic metric. The problem for finding top-k flows is shown to be in PTIME, NP-complete or undecidable, depending on memory bound for partial flows preceding a given choice. It is quite different from the aggregation problem studied in this work. Indeed, (a) [30,31] focus on aggregate values determined by monotonic functions on weights or likelihoods that are predefined for each edge in a flow, computed when traversing a flow. In contrast, we consider synthesis of aggregate values upward from multiple processes that run in parallel. Our aggregate values can only be computed after a complete run, and are determined by the data flow: the output artifact of one service is the input of another. Hence the complexity results of [30,31] do not carry over to our setting, and vice versa. (b) This work provides the complexity bounds of the aggregation problem for mediators with various underlying structures: cyclic graph, DAG or tree, which are not studied in [30,31]. (c) This work provides the first complexity bounds of the aggregation and termination problems, for mediators which nondeterministically select transition and synthesis rules. It should be remarked that a mediator in the model of [30,31] is specified in terms of a set of recursively defined DAGs, in which each node (activity) has multiple implementations that can be picked at run-time. While the nondeterministic selection of implementations for activities and the termination of the evaluation algorithms are discussed in [30,31], their setting is quite different from ours and moreover, no complexity bound is given there.

A number of standards have been developed for specifying Web services, such as WSDL [32], WSCL [33], OWL-S [34], SWFL [35] and BPEL [36]. A variety of models have also been proposed to characterize services supported by those standards, based on finite-state automata [1–3], data-driven transducers [4–9] or artifacts [10–12]. The notion of SWMs is a refinement of synthesized mediators studied in [8], which shows that its mediators are able to express automaton and transducer abstractions of services. SWMs refine mediators of [8] by defining synthesis rules in terms of aggregation functions. They emphasize data flow among component services, along the same lines as [17]. Meanwhile SWMs specify control flow in terms of transitions of a transducer. To our knowledge, (a) only [8] and the split–join operator of OWL-S [34] allow one to express synthesis operations, and (b) no previous model supports aggregation syntheses.

As remarked earlier, several verification problems have been investigated for Web services [2,4,6–9,11,14–17]. Complexity bounds have also been developed for the composition problem [1,3,5,8,17]. To the best of our knowledge, however, no previous work has studied the aggregation problem. In particular, the aggregation problem is quite different from the composition problem. The latter is a decision problem to determine whether there exists a mediator that coordinates available services to deliver a requested service; in contrast, the former is an optimization problem that aims to find a realization of a *given* mediator to maximize (or minimize) certain values in an artifact.

An artifact is an identifiable record in which attributes may be created, updated, or deleted [11,10,13,21]. It represents the life-cycle and business-relevant data of a business entity [12]. In this work we use artifacts to characterize input messages to a composite service, communications between components during a run of the service, and the output of the run.

Organization. We present SWMs in Section 2, and formulate the aggregation problem in Section 3. We establish the undecidability of the problem in Section 4. We identify decidable cases of the aggregation problem, and provide their matching complexity bounds in Section 5. In Section 6 we introduce NSWMs, investigate their aggregation analysis, and study the termination problem for SWMs and NSWMs. Section 7 summarizes the main results and identifies open problems.

2. Synthesized mediators

In this section we define the syntax and the semantics of SWMs.

2.1. Synthesized mediators

Before we formally define SWMs, we first describe artifacts and component templates.

Artifacts and templates. SWMs will be based on artifacts, which we define, following [12], as records specified by an *artifact schema*

$$R_A = (\mathsf{val} : \mathbb{Q}, A_1 : \theta_1, \ldots, A_n : \theta_n),$$

where each A_i is an attribute and θ_i is its domain. We have a designated attribute val with the domain $\mathbb{Q}$ of rational numbers (for storing aggregate values). We assume that a special symbol $\perp$ is in each of the domains, denoting undefined as usual. We use $\mathcal{I}(R_A)$ to denote the set of all artifacts of schema R_A.

We assume a countably infinite set Γ of *template names* for component services, ranged over by τ. Each template denotes a service of a certain functionality.

Mediators. A synthesized mediator (SWM) is a deterministic finite-state transducer defined in terms of component templates. When the templates are realized with available services, the SWM coordinates those services to deliver a requested composite service. More specifically, upon receiving an artifact, the SWM invokes the component services to operate on the artifact, and redirects it by routing the output of one service to the input of another [36]. It generates the output of the requested service by synthesizing certain values in the artifacts updated by the component services.

Definition 2.1. A *synthesized mediator* (for Web services, referred to as an SWM) over an artifact schema R_A is defined as $M = (Q, \delta, \sigma, q_0)$, where Q is a finite set of *states*, q_0 is the *start state*, δ is a set of *transition rules*, and σ is a set of *synthesis rules*, such that for each state $q \in Q$, there exist a unique transition rule $\delta(q)$ and a unique synthesis rule $\sigma(q)$:

$$\delta(q): \quad (q, \phi) \to (q_1, \tau_1), \ldots, (q_k, \tau_k),$$

$$\sigma(q): \quad \mathsf{val}(q) \leftarrow F_q\big(\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k)\big).$$

Here $q, q_1, \ldots, q_k$ refer to states in Q, and

- for each $i \in [1, k]$, τ_i is a template name from Γ, referred to as a *component template* of M; the set of all the templates of M is denoted by $\Gamma(M)$;
- ϕ, called the *precondition* of q, is a PTIME-computable predicate over artifacts of schema R_A;
- $k \geqslant 0$; in particular, when $k = 0$, the right-hand side (RHS) of the rules $\delta(q)$ and $\sigma(q)$ are empty; and
- $F_q : \mathbb{Q}^k \to \mathbb{Q}$ is a PTIME-computable function, referred as a *synthesis function*, and $\mathsf{val}(q)$ is the aggregate value in state q computed by $F_q()$.

For a transition $(q, \phi) \to (q_1, \tau_1), \ldots, (q_k, \tau_k)$, we call $q_1, \ldots, q_k$ the *successor* states of q carrying templates $\tau_1, \ldots, \tau_k$, respectively.

Example 2. The mediator M_1 described in Example 1 can be expressed as an SWM. The artifact schema for mediator M_1 consists of attributes specifying (1) departure city, travel dates, and the number of tickets, (2) a list T_l of free time slots to be filled, (3) a list A_l of activities, initially empty, and (4) val indicating the total cost of a trip, initially $\perp$. We define mediator $M_1 = (Q_1, \delta_1, \sigma_1, q_1)$, where $Q_1 = \{q_1, q_s, q_c, q_f, q_h, q_a, q_r, q_p, q_l\}$, and q_1 is the start state. The transition rules δ_1 and synthesis rules σ_1 are shown in Fig. 2.

In mediator M_1, $\Gamma(M_1)$ includes templates τ_f, τ_h, τ_a, τ_p and τ_l. As shown in Fig. 1, these templates are to be realized with available services for checking *flight*, *hotel*, *activity*, *cruise package* and *logging*, respectively. Each of these services updates certain attribute values of the artifact. For example, τ_a updates attributes A_l and T_l by filling a time slot with an activity. In addition, $\Gamma(M_1)$ contains a dummy template τ_{id}, which simply passes artifact to its successor state without incurring any changes.

Note that the synthesis rule for q_1 is defined with aggregation operator min, while the synthesis rule for q_s is defined in terms of the sum aggregate. We shall explain the semantics of mediator M_1 in Example 3.

$$
\begin{array}{llll}
(q_1, \text{true}) \rightarrow (q_s, \tau_{id}), (q_c, \tau_{id}) & & \text{val}(q_1) \leftarrow \min(\text{val}(q_s), \text{val}(q_c)) & \\
(q_s, \text{true}) \rightarrow (q_f, \tau_f), (q_h, \tau_h), (q_a, \tau_a) & & \text{val}(q_s) \leftarrow \text{val}(q_f) + \text{val}(q_h) + \text{val}(q_a) & \\
(q_a, \phi_a) \rightarrow (q_a, \tau_a), (q_r, \tau_{id}) & /^{*}\ \phi_a \text{ is } t.T_l \neq \emptyset\ ^{*}/ & \text{val}(q_a) \leftarrow \text{val}(q_a) + \text{val}(q_r) & \\
(q_c, \text{true}) \rightarrow (q_a, \tau_a), (q_p, \tau_p) & & \text{val}(q_c) \leftarrow \text{val}(q_a) + \text{val}(q_p) & \\
(q_p, \text{true}) \rightarrow (q_r, \tau_{id}), (q_l, \tau_l) & & \text{val}(q_p) \leftarrow \text{val}(q_r) + \text{val}(q_l) & \\
(q_f, \text{true}) \rightarrow . & & \text{val}(q_f) \leftarrow . & /^{*} \text{ similarly for } q_h, q_l, q_r\ ^{*}/
\end{array}
$$

Fig. 2. The transition rules and synthesis rules of mediator M_1.

2.2. Semantics of mediators

The semantics is defined via *realizations* of SWMs, which substitute available library services for template names. Once this is done, we give two ways to present the semantics of SWMs: a traditional, purely operational one, and an equivalent semantics that describe the run at once, rather than via a sequence of steps.

Realizing SWMs. We view available services as functions on artifacts, i.e., functions $f : \mathcal{I}(R_A) \rightarrow \mathcal{I}(R_A)$. We only impose a condition that such functions be tractable, i.e., PTIME-computable. We assume there is a *library* L of available services to choose from. The library can be built by leveraging techniques for Web service discovery (e.g., [37,38]).

In a nutshell, the output of an available service is used to update attribute values of the input artifact. The service conducts the computation based on data in its local database and the input artifact. While in practice it may take additional input from the users, to simplify the discussion we assume that all the input parameters are encompassed in the input artifact as attributes. This assumption does not change the complexity bounds for the aggregation problem to be investigated.

To make a composite service, an SWM needs to be realized by substituting available library services for its templates. Thus, we define a *realization* of an SWM M in library L as a mapping $\rho : \Gamma(M) \rightarrow L$, from the set $\Gamma(M)$ of templates of M to L. We denote the result of substituting a library service $\rho(\tau)$ for each occurrence of τ in M by $M[\rho]$, referred to as the *composite service of M realized by ρ*.

To ensure that the services realized make sense, we need to impose constraints on realizations. For instance, it is not sensible if one realizes a template intended for airfare with a service for hotel. Thus, we define a *realization constraint* as a mapping $\lambda : \Gamma(M) \rightarrow \mathcal{P}(L)$, from $\Gamma(M)$ to the powerset $\mathcal{P}(L)$ of L. That is, a template τ is restricted to a set $\lambda(\tau)$ of available services that have the required functionality, such that τ is only allowed to be realized with a service in the subset $\lambda(\tau)$ of L. That is, realization constraints classify services in the library based on their functionality.

A realization ρ of M is said to be *valid w.r.t.* realization constraint λ if for each τ in $\Gamma(M)$, we have $\rho(\tau) \in \lambda(\tau)$. We also say that a realization constraint λ is *deterministic* if it uniquely determines the library service for each template, i.e. $|\lambda(\tau)| = 1$ for all $\tau \in \Gamma(M)$.

Operational semantics. It will be defined in terms of runs of composite services. A composite service $M[\rho]$, where M is defined over an artifact schema R_A, runs on artifacts of R_A. We present two equivalent notions of a run: one of purely operational, and the other of more denotational flavor.

For the operational notion, we define a step relation $\Rightarrow_{(M[\rho],t_0)}$, where t_0 is an artifact. The relation is between *execution trees* [5,1]. One starts with a single-node execution tree labeled with the triple $(q_0, t_0, \perp)$, and proceeds until a terminal execution tree is reached, on which the step relation is not applicable. Then the value of the third attribute of the root's label in that execution tree is the result of running the composite service, i.e., $M[\rho](t_0)$.

More precisely, in an execution tree, each node v is labeled with a triple (q, t, w), where $q \in Q$, t is an artifact of R_A, and $w \in \mathbb{Q} \cup \{\perp\}$. We refer to the value w as $\text{val}(v)$. For two execution trees ξ and ξ', we write $\xi \Rightarrow_{(M[\rho],t_0)} \xi'$ if one of the following conditions holds.

Spawning. If there exists a leaf node v of ξ labeled with $(q, t, \perp)$ (where the transition rule for q is $(q, \phi) \rightarrow (q_1, \tau_1), \ldots, (q_k, \tau_k)$) and ξ' is obtained from ξ as follows.

- If either $k = 0$ or ϕ evaluates to false on t (i.e., either q has no successor state, or the precondition for q does not hold), then ξ' is obtained from ξ by setting $\text{val}(v)$ to the value of the val attribute of t.
- Otherwise ξ' is obtained from ξ by spawning k children $u_1, \ldots, u_k$ of v, in parallel. For each $i \in [1, k]$, a distinct node u_i is created as the i-th child of v. The node u_i is labeled with $(q_i, \rho(\tau_i)(t), \perp)$, i.e., it invokes available service $\rho(\tau_i)$ and labels u_i with the updated artifact $\rho(\tau_i)(t)$.

Synthesizing. If there is no leaf node to which a transition rule applies, then ξ' is obtained from ξ by picking a node v labeled by $(q, t, \perp)$ so that none of its successors $u_1, \ldots, u_k$ has $\text{val}(u_i) = \perp$, and updating $\text{val}(v)$ according to the synthesis rule: $\text{val}(v)$ gets the value $F_q(\text{val}(u_1), \ldots, \text{val}(u_k))$, where F_q is the aggregate from the synthesis rule for q.

In other words, the synthesis rule is applied if $\text{val}(v) = \perp$ as soon as $\text{val}(u_i)$ is available for all $i \in [1, k]$.

The run starts from an execution tree ξ_0 consisting of a single root node r, labeled with q_0, the input artifact t_0 and carrying $\text{val}(r) = \perp$. Then an execution tree is generated top-down; spawning new nodes stops at a node reached if either it

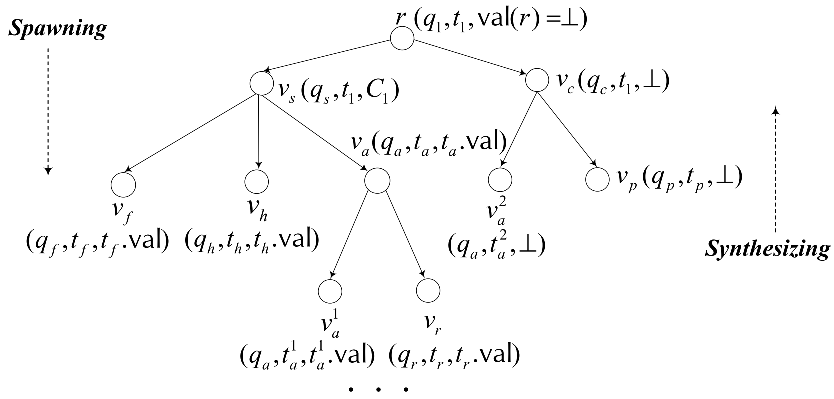

Fig. 3. An example execution tree.

is in a "final state" q indicated by the transition rule of q (with an empty RHS), or its precondition ϕ is not satisfied. In both cases val at such a node carries a non-$\perp$ value. The synthesis rule for state q is applied bottom-up to a node v labeled with $(q, t, \perp)$ as soon as $\mathsf{val}(u_i)$'s are available for all the children of v.

If the process stops, $\mathsf{val}(r)$ is the output. More precisely, the *result* of the run of $M[\rho]$ on artifact t_0 is an execution tree ξ such that $\xi_0 \Rightarrow^* \xi$ and there is no distinct ξ' such that $\xi \Rightarrow \xi'$ (here $\Rightarrow^*$ is the reflexive-transitive closure of $\Rightarrow$). The *output* $M[\rho](t_0)$ is the content of $\mathsf{val}(r)$ at the root r of the result of the run.

The process may not necessarily stop when a mediator M is "recursively defined", i.e., when a state in M can reach itself after one or more transitions. In other words, there may not exist a finite execution tree ξ such that $\xi_0 \Rightarrow^* \xi$ and ξ cannot be further expanded via spawning. When this happens, $M[\rho](t_0)$ is undefined. We shall study termination analysis in Section 6.

Denotational semantics. Note that while there may be multiple runs of a composite service, their results coincide, and thus the output is uniquely defined. In fact, one can compactly represent the output of such runs by a single tree, as shown in the easily verified proposition below. The proposition suggests a semantics of denotational flavor, which is equivalent to its operational counterpart given above.

Proposition 1. *For a composite service $M[\rho]$ and an artifact t_0 of schema R_A, the result of a run and the output of $M[\rho]$ on t_0 are either a $(Q \times I(R_A) \times \mathbb{Q})$-labeled tree ξ and a number $w_0 \in \mathbb{Q}$ satisfying the following conditions*:

1. *the root of ξ is labeled with (q_0, t_0, w_0)*;
2. *consider a node v of ξ labeled with (q, t, w), where $(q, \phi) \to (q_1, \tau_1), \ldots, (q_k, \tau_k)$ is the transition rule for q*;
   (a) *v is a leaf iff $w = t.\mathsf{val}$ and either $\phi(t) = false$ or $k = 0$*;
   (b) *v is a non-leaf node iff it has k children labeled with $(q_i, \rho(\tau_i)(t), w_i)$ for $i \in [1, k]$ so that $w = F_q(w_1, \ldots, w_k)$, where F_q is the aggregate in the synthesis rule for q*;

or are undefined.

Example 3. Recall mediator M_1 from Example 1. Given an artifact t_1 of schema R_1 and a realization ρ_1, the execution tree specifying the run of $M_1[\rho_1]$ on t_1 is constructed as follows, as depicted in Fig. 3.

(1) It starts with a tree ξ_0 consisting of only the root node r, labeled with $(q_1, t_1, \mathsf{val}(r) = \perp)$.

(2) Since the preconditions for q_s and q_c are true, the tree ξ_0 is expanded to ξ_1 by creating two children v_s and v_c for root r, labeled with $(q_s, t_1, \perp)$ and $(q_c, t_1, \perp)$, respectively. Note that the dummy service τ_{id} simply passes the input artifact t_1 to v_s and v_c.

(3) At node v_s, the available services $\rho_1(\tau_f)$, $\rho_1(\tau_h)$ and $\rho_1(\tau_a)$ are invoked unconditionally, in parallel with parameter t_1 associated with v_s. The tree ξ_1 is expanded by creating three children v_f, v_h, v_a for v_s.

At node v_f, assume that t_f is the output artifact of $\rho_1(\tau_f)$, and $t_f.\mathsf{val}$ is the airfare found by $\rho_1(\tau_f)$ based on the data in the input artifact t_1 and the local database of $\rho_1(\tau_f)$. Since state q_f does not have any successor state, v_f does not spawn any new node, and $\mathsf{val}(v_f)$ is simply set to be $t_f.\mathsf{val}$; similarly for v_h.

On the other hand, at node v_a, if the precondition $\phi_a(t_1)$ is satisfied, service $\rho_1(\tau_a)$ is triggered to find an activity. It returns an artifact t_a, which updates t_1 by filling a free time slot with an activity, i.e., adding the newly chosen activity to $t_1.A_l$ (treated as $t_a.A_l$), and removing the corresponding slot from $t_1.T_l$ (denoted as $t_a.T_l$). It spawns two children v_a^1 and v_r for v_a, and passes t_a to them. While the node v_r simply retains $t_a.\mathsf{val}$ for synthesizing (denoted as $t_r.\mathsf{val}$), the process repeats at node v_a^1, which invokes $\rho_1(\tau_a)$ to select activities for the remaining time slots in $t_a.T_l$. The tree expands until all the free time slots are filled, i.e., when the precondition ϕ_a no longer holds.

(4) As soon as the spawning process terminates for the subtree of v_a, the synthesizing phase starts for the subtree of v_s. Synthesizing val values upwards, $\mathsf{val}(v_a)$ is set to be the sum of the costs for all the chosen activities. When $\mathsf{val}(v_a)$ is available, $\mathsf{val}(v_f) + \mathsf{val}(v_h) + \mathsf{val}(v_a)$ is computed and assigned as the value of $\mathsf{val}(v_s)$, which is the cost C_1 as shown in Fig. 3.

(5) Similarly, at node v_c two children v_a^2 and v_p are created. In particular, at node v_p, service $\rho_1(\tau_p)$ is triggered to select a cruise package, which yields artifact t_p. Based on the package selected and its constraint on logging, a hotel is chosen by invoking service $\rho_1(\tau_l)$, which takes t_p as the input parameter.

Along the same lines as described above, the subtree rooted at v_c is completed and $\mathsf{val}(v_c)$ is computed. At this point $\mathsf{val}(r)$ can be computed, as $\min(\mathsf{val}(v_s), \mathsf{val}(v_c))$. This yields the result of the run, an execution tree in which no node v is labeled with $\mathsf{val}(v) = \perp$. The output $M_1[\rho_1](t_1)$ of the run is $\mathsf{val}(r)$.

To sum up, a transition rule indicates a business rule, and the precondition for each state determines whether its associated business rule should be carried out or not. An SWM specifies the control flow in terms of its transition rules, and the data flow with artifacts. There exist dependencies on the artifacts, e.g., the output artifact of $\rho_1(\tau_p)$ is the input of $\rho_1(\tau_l)$ in the example above; that is, the choice of hotel depends on what cruise package is selected in the previous state, as various cruise packages impose different lodging constraints. Also, to simplify the discussion, we only take a single artifact as input and produce a single value val as output. However, the definition of SWMs can be readily extended such that a composite service may take multiple artifacts as input and return multiple artifacts as output (including but not limited to val), and this does not change the results in the paper.

3. The aggregation problem

We now present the aggregation problem. Given an SWM M over artifact schema R_A, an artifact t of R_A, a library L of available services and a realization constraint λ, *the aggregation problem* is to find a realization ρ of M in L that is valid *w.r.t.* λ and maximizes (or minimizes) $M[\rho](t)$, the output of M on t.

Intuitively, given an input t and a mediator M, the aggregation synthesis is to generate a composite service "on-the-fly" [39] that is "optimal" for user's request, by realizing templates of M with available services *w.r.t.* the user's input. For instance, the aggregation synthesis for SWM M_1 of Example 1 is an instance of the aggregation (minimization) problem.

To study the complexity, we turn to a decision version of the problem. In such a version, we are interested in a valid realization ρ of M in L so that $M[\rho](t) \geqslant B$, for a predefined bound B.

PROBLEM:	$\mathsf{AGP}(M, L, \lambda, t)$
INPUT:	1. an SWM M on artifact schema R_A; 2. an artifact t of R_A; 3. a library L of available services; 4. a realization constraint λ.
QUESTION:	Does there exist a realization ρ of M in L valid *w.r.t.* λ so that $M[\rho](t) \geqslant B$?

One can change the sign of all the values and aggregate functions and arrive at an equivalent minimization problem which asks whether $M[\rho](t) \leqslant B$. If we want to emphasize whether we refer to the maximization ($M[\rho](t) \geqslant B$) or minimization ($M[\rho](t) \leqslant B$) version, we shall write $\mathsf{AGP}_{\mathsf{max}}$ or $\mathsf{AGP}_{\mathsf{min}}$, respectively.

Our goal is to investigate the complexity bounds of $\mathsf{AGP}(M, L, \lambda, t)$ for SWMs M of various structures. More precisely, we define the *mediator graph* of an SWM $M = (Q, \delta, \sigma, q_0)$, denoted by $G[M]$, as a directed edge-labeled graph $G[M] = (Q, E, L)$ in which there is an edge (q, q') in E labeled with τ if q' is a successor state of q carrying template τ, i.e., (q', τ) is in the RHS of the transition rule for q in M. In the sequel we simply write $G[M]$ as (Q, E) when L is clear from the context. An SWM M is recursively defined if $G[M]$ is cyclic.

We start by showing that the general problem is undecidable even for very simple SWMs that have a single state and whose underlying graph is a self-loop. In fact, we show that for *every* graph containing a cycle, the aggregation problem for mediators with that underlying graph is undecidable (even if some of the parameters are fixed). As an example, Fig. 1 depicts an SWM with a cyclic graph structure.

So this suggests a restriction to SWMs whose mediator graph is a DAG. We shall show that for such SWMs the problem is decidable in PSPACE, and the further restriction to tree-structured SWMs puts the problem in NP.

4. Aggregation synthesis: undecidability

In this section we show that the general problem $\mathsf{AGP}(M, L, \lambda, t)$ is undecidable, and identify restrictions that need to be put on the parameters of the problem to achieve decidability.

Recall that the mediator graph for an SWM M is the graph $G[M]$ whose nodes are reachable states of M, and which has an edge from q to q' if q' appears in the right-hand side of the unique transition rule for q in M.

We then have the following undecidability result. Recall that a realization constraint λ is *deterministic* if $|\lambda(\tau)| = 1$ for all $\tau \in \Gamma(M)$, i.e., for each template, the library service realizing it is uniquely determined.

Theorem 2. *Let G be an arbitrary connected graph with a cycle. Then there exists an SWM M_0 whose mediator graph is G, a fixed library L_0 and a deterministic realization constraint λ_0 such that the problem* $\mathsf{AGP}(M_0, L_0, \lambda_0, t)$, *whose only input is the artifact t, is undecidable.*

Proof. For now assume that G is the simplest possible graph with only one node and a cycle (i.e., it has only one state q and one self-loop (q, q)).

We show the undecidability by reduction from the existence of solutions of Diophantine equations with fixed-degree and fixed number of variables. It is known that one can fix numbers d and k so that the following problem is undecidable: given a polynomial $p(x_1, \dots, x_k)$ with integer coefficients of degree at most d, does it have an integer solution? That is, do there exist integers $j_1, \dots, j_k$ so that $p(j_1, \dots, j_k) = 0$. This was shown in [40], and the bounds on k and d have since been improved, for example, to $d = 4$, $k = 58$ or $d = 16$, $k = 29$, see [41].

Let $\delta_1, \dots, \delta_m$ enumerate all the tuples of integers $(n_1, \dots, n_k)$ so that $\sum_i n_i \leqslant d$. These correspond to the monomials $x_1^{n_1} \dots x_k^{n_k}$. The artifact schema R_A contains attributes $A_1, \dots, A_m$, and an extra attribute I. The idea is that the values of that attribute will be iterated, while looking for a (code of a) solution. Note that since d and k are fixed, the number m is bounded by a constant and can be considered to be fixed as well.

Suppose that we are given as an input a Diophantine polynomial

$$p(x_1, \dots, x_k) = \sum_{i=1}^{m} a_i x_1^{\delta_i(1)} \dots x_k^{\delta_i(k)},$$

where for $\delta_i = (n_1, \dots, n_k)$, we denote n_j by $\delta_i(j)$. We represent it as an artifact t_p, where val is set to 0, each A_i is set to the corresponding coefficient a_i, and I is set to 0.

We construct SWM M_0 as follows. It keeps iterating the value of I, viewing it as a code of a k-tuple of natural numbers. Since k is fixed, this value can be decoded into a k-tuple in polynomial time. We use a fixed library consisting of a single function f that increases the value of I by 1. Then, in state q, the SWM M_0 decodes the code value and computes the value of the polynomial. If the value is 0, the val attribute is set to 0 and propagated up. Otherwise the function f is invoked to increase the code by 1, and the process proceeds.

We now define this formally. Assume that $decode_k : \mathbb{N} \to \mathbb{N}^k$ is a polynomial-time computable function that decodes a number into a k-tuple. This can be obtained by iterating the standard coding of pairs, i.e., a one-to-one function $pair : \mathbb{N}^2 \to \mathbb{N}$. Then the coding of $(n_1, \dots, n_k)$ is $pair(n_1, pair(n_2, \dots, pair(n_{k-1}, n_k) \dots))$. Since the function $pair$ and the corresponding decoding are polynomial-time computable, and k is fixed, then $decode_k$ is computable in polynomial time. The library L_0 has only one function f which increases the value of the I attribute by 1. SWM M_0 will have only one template τ, and the deterministic realization constraint is $\lambda_0(\tau) = \{f\}$. The transition rule of q is:

$$(q, P \neq 0) \to (q, \tau).$$

Here P takes the value N of the I attribute and computes $p(decode_k(N))$. Since all the coefficients of p are present in the artifact and the degree is constant, the computation takes polynomial time.

The synthesis rule is simply $\mathsf{val}(q) \leftarrow \mathsf{val}(q)$, i.e., the value is propagated all the way to the root. It is now routine to verify that for p having an integer solution, M_0 will return 0, and for p not having a solution it will not terminate.

The graph of M_0 is G, which has one node with a self-loop. If we have an arbitrary graph with a cycle $q_0 \to q_1 \to \dots \to q_{l-1} \to q_0$, we simply modify the transition rule so that it cycles through these states, i.e., the rules are $(q_i, P \neq 0) \to (q_j, \tau)$, where $j = (i + 1) \bmod l$, with q_0 being the initial state. Then the preceding proof applies verbatim. This concludes the proof. $\square$

A slight modification of the proof shows the following undecidability result.

Corollary 3. *The aggregation problem is undecidable even if the library, the (deterministic) realization constraint, the artifact, and the cyclic mediator graph are fixed. That is, for an arbitrary connected graph G with a cycle, there exist a fixed library L_0, a deterministic realization constraint λ_0 and an artifact t_0 so that the problem* $\mathsf{AGP}(M, L_0, \lambda_0, t_0)$, *whose only input is an SWM M with the mediator graph G, is undecidable.*

Proof. We follow the previous proof and remove the coefficients of the polynomial from the artifact, and instead put attributes for the values of the decoded tuple of variables. The SWM produces the tuple from a code and computes the polynomial in the precondition, i.e., follows a transition rule $(q, p(\bar{n}) \neq 0) \to (q, \tau)$, where p is the Diophantine polynomial and $\bar{n}$ is the k-tuple holding the decoded values. The library contains a single function f as in the previous proof. $\square$

Analyzing the proof, we see that there are two main reasons for undecidability:

1. cyclicity of the mediator graph (even a single cycle leads to undecidability), and
2. the infinite domain of attribute values of the artifact.

The second constraint is essential for many applications as artifacts store numbers, dates, strings, etc. So we need to impose restrictions on the mediator graph. As no cycles are allowed, we shall look at mediator graphs which are DAGs and trees in the next section. But now, for completeness only, we present a simple result for the case of fixed-size domain.

Proposition 4. *When the size of the domain of each attribute of the artifact schema is fixed,* $\mathsf{AGP}(M, L, \lambda, t)$ *can be solved in single-exponential time. If M and the artifact schema are fixed as well, it is solvable in polynomial time.*

Proof. Let θ be an upper bound on the size of the domain. If the artifact schema contains m attributes, we can have at most θ^m possible values of artifact values. Suppose $\tau_1, \dots, \tau_k$ are the templates used in SWM M, and L is the library. The library is finite (since there are finitely many possible artifacts) and there are at most $|L|^k$ realizations ρ of M. For each such ρ, we can represent $M[\rho]$ as a tree automaton $A_{M[\rho]}$ whose states are of the form (q, t'), where q is a state of M and t' is a possible value of the artifact tuple. The transitions of $A_{M[\rho]}$ ensure that both transition and synthesis rules of M are respected. It is clear that the set of states of the automaton is exponential in the size of the input of $\mathsf{AGP}(M, L, \lambda, t)$, and that the transitions can be computed in single-exponential time. Finally, the accepting states are (q_0, t'), where val of t' is at least B. Then $A_{M[\rho]}$ accepts a tree if and only if $M[\rho](t) \geqslant B$.

Now for each ρ, we construct $A_{M[\rho]}$ and test it for nonemptiness; since the latter takes polynomial time in the size of the automata, and there are exponentially many ρ's, the algorithm runs in exponential time. The answer to $\mathsf{AGP}(M, L, \lambda, t)$ is true if and only if the language of one of the $A_{M[\rho]}$'s is nonempty.

Finally, if m (the number of attributes) is fixed, then there exists a fixed number of artifacts, and if M is fixed, then there are polynomially many realizations ρ; in this case, the algorithm runs in polynomial time. $\square$

5. Decidable cases

In this section we identify special decidable cases of the aggregation problem. We study $\mathsf{AGP}(M, L, \lambda, t)$ for SWMs M that are not recursively defined, i.e., when the mediator graph $G(M)$ of M is acyclic. As a result, one does not have to worry about the termination of runs of composite services realized with these SWMs.

5.1. Tree-structured mediators

We start with $\mathsf{AGP}(M, L, \lambda, t)$ for *tree-structured* SWMs M, i.e., when $G(M)$ is a tree.

Complexity. Our first result shows that the aggregation problem indeed becomes decidable when $G(M)$ is a tree. In fact, it can be solved in single-exponential time, which is acceptable for static analysis of specifications such as SWMs.

The problem is, however, intractable even for simple "pipelined" SWMs, i.e., when $G(M)$ has a linear (chain) structure. More specifically, we say that M has a *pipelined structure* if every transition rule of M either has an empty right-hand side, or is of the form $(q, \phi) \to (q', \tau)$. Moreover, the intractability is rather robust: it holds even if we fix the library L (which is a reasonable assumption, as in practice, a library of available services may be relatively stable: it is only updated periodically).

Theorem 5. $\mathsf{AGP}(M, L, \lambda, t)$ *is NP-complete for tree-structured SWMs. The problem remains NP-hard when the library L is fixed and when the mediator M has a pipelined structure.*

Proof. We show that $\mathsf{AGP}_{\mathsf{max}}(M, L, \lambda, t)$ and $\mathsf{AGP}_{\mathsf{min}}(M, L, \lambda, t)$ are NP-complete, and are already NP-hard even when L is fixed and when M has a pipelined structure.

(1) $\mathsf{AGP}_{\mathsf{max}}(M, L, \lambda, t)$. Given M, L, λ, t and a number B, we show that it is NP-complete to determine whether there exists a realization ρ such that ρ is valid *w.r.t.* λ and $M[\rho](t) \geqslant B$.

Upper bound. We first show that for any tree-structured M, the problem is in NP, by giving an NP algorithm for deciding whether there exists such a realization ρ. The algorithm takes two steps: it first guesses ρ, and then checks whether ρ is valid *w.r.t.* λ and $M[\rho](\tau) \geqslant B$. The checking can obviously be done in PTIME, since (a) for all templates τ in M, checking whether $\rho(\tau) \in \lambda(\tau)$ is in PTIME, (b) $M[\rho](\tau)$ can be computed in PTIME because all preconditions and synthesis rules are defined with PTIME-computable functions, and moreover, for all templates τ in M, $\rho(\tau)$ is a PTIME function in the library L. Hence when M has a tree structure, the problem is in NP.

Lower bound. We next show that the aggregation problem $\mathsf{AGP}_{\max}(M, L, \lambda, t)$ is NP-hard when L is fixed and M has a pipelined structure, by reduction from 3SAT. It is known that 3SAT is NP-complete (cf. [42]). Given an instance φ of 3SAT, we construct a pipelined SWM M, a library L of available services, a realization constraint λ, an initial artifact t and a number B, such that φ is satisfiable iff there exists a realization ρ that is valid *w.r.t.* λ and moreover, makes $M[\rho](t) \geqslant B$.

Assume that $\varphi = C_1 \wedge \cdots \wedge C_n$, defined with variables $x_1, \ldots, x_m$, where for each $i \in [1, n]$, C_i is a clause of the form $l_1 \vee l_2 \vee l_3$, and l_i is either a variable x_j or its negation $\bar{x}_j$. We construct R_φ, L, M and λ as follows.

(A) The artifact schema R_φ is defined to be (X, val), where X is to hold a binary number $b_1 \ldots b_m$, encoding a truth assignment for $x_1, \ldots, x_m$, and val is to denote the truth value of φ. The initial artifact t is $(X = 0,\ \mathsf{val} = 0)$.

(B) The library L consists of three services: f_T, f_F and f_1, where (a) f_T takes $(X = b_1 \ldots b_j, \mathsf{val})$ as input, and returns $t = (X = b_1 \ldots b_j 1, \mathsf{val})$, i.e., by adding 1 as the last digit of the updated $t.X$, (b) similarly, f_F expands $t.X$ by adding 0 as the last digit of $t.X$, and (c) f_1 is a constant function that returns 1. Note that L is fixed: it is independent of φ.

(C) The SWM M is defined as (Q, δ, σ, q_1), where the set of states $Q = \{q_j \mid j \in [1, m+2]\}$, and the rules δ and σ are given as follows:

$$(q_j, \mathsf{true}) \to (q_{j+1}, \tau_j),$$
$$\mathsf{val}(q_j) \leftarrow \mathsf{val}(q_{j+1}) \quad /^{*} \text{ for } j \in [1, m] \ ^{*}/$$

$$(q_{m+1}, \phi) \to (q_{m+2}, \tau_v),$$
$$\mathsf{val}(q_{m+1}) \leftarrow \mathsf{val}(q_{m+2})$$

$$(q_{m+2}, \mathsf{true}) \to .$$
$$\mathsf{val}(q_{m+2}) \leftarrow .$$

Here ϕ is a Boolean function $(C_1 \wedge \cdots \wedge C_n)$ $[x_1/t.X[1]] \ldots [x_m/t.X[m]]$, where $t.X[j]$ denotes the j-th digit of $t.X$. The set $\Gamma(M)$ of templates consists of τ_j for $j \in [1, m]$ and τ_v.

Intuitively, δ and σ specify a control flow of a pipelined structure that generates a truth assignment for variables of φ, step by step. Specifically, in state q_j for $j \leqslant m$, the truth value of x_j is added as the last digit of $t.X$, by invoking either f_T or f_F in the library L. In state q_{m+1}, the precondition p evaluates the truth value of φ based on the last m digits of the truth assignment $t.X$, in PTIME; if the condition is satisfied, $t.\mathsf{val}$ is changed to 1 by invoking f_1; otherwise $t.\mathsf{val}$ remains to be 0.

The constraint λ is defined as follows: $\lambda(\tau_j) = \{f_T, f_F\}$ for $j \in [1, m]$, and $\lambda(\tau_v) = \{f_1\}$.

(D) The constant B is set to be 1.

We show that the construction given above is indeed a reduction. Assume that φ is satisfiable. Then there exists a truth assignment μ for variables in φ that satisfies φ. Define a realization ρ such that for all $j \in [1, m]$, $\rho(\tau_j) = f_T$ if $\mu(x_j) = 1$, and $\rho(\tau_j) = f_F$ otherwise. Obviously $M[\rho](t) = 1$, i.e., $M[\rho](t) \geqslant 1$.

Conversely, suppose that there exists a realization ρ such that $M[\rho](t) \geqslant 1$. Define a truth assignment μ for φ such that for all $j \in [1, m]$, $\mu(x_j) = 1$ if $\rho(\tau_j) = f_T$ and $\mu(x_j) = 0$ otherwise. Then μ satisfies φ.

(2) $\mathsf{AGP}_{\min}(M, L, \lambda, t)$. The proof for the upper bound is by giving an NP algorithm. The algorithm is the same as its counterpart for $\mathsf{AGP}_{\max}(M, L, \lambda, t)$, except that the last step of the algorithm inspects whether $M[\rho](t) \leqslant B$.

The proof for the lower bound is by reduction from non-tautology, which is NP-complete (cf. [42]). The definitions of M, L, λ, t are the same as the construction given in (1), except the following: (a) $B = 0$, (b) $t.\mathsf{val} = 1$ in the initial artifact, (c) the service f_1 is a constant function that returns 0, and (d) the precondition p tests whether the given instance of the non-tautology problem evaluates to false. $\square$

In light of this intractability result one might be tempted to develop a PTIME approximation algorithm for the aggregation problem such that one can still efficiently find a solution with certain performance guarantee. However, this is also infeasible. The result below shows that the aggregation problem is not even in APX (see, e.g., [43]), the class of problems that allow PTIME approximation algorithms with approximation ratio bounded by a constant.

We show a stronger result: $\mathsf{AGP}(M, L, \lambda, t)$ does not even allow any PTIME approximation algorithms with approximation ratio bounded by any polynomial. Following [43], we say that an algorithm *achieves a polynomial approximation ratio* n^l for a maximization (resp. minimization) problem if for every instance of the problem, it produces a solution of value at least $\frac{1}{1+n^l}\mathsf{OPT}$, i.e., in the range $[\frac{1}{1+n^l}\mathsf{OPT}, \mathsf{OPT}]$ (resp. at most $(1+n^l)\mathsf{OPT}$), where l is fixed and OPT is the value of the optimal solution. We refer to such an algorithm as a n^l*-approximation algorithm*. The result below tells us that no matter what n^l is used, it is impossible to find a PTIME n^l-approximation algorithm for $\mathsf{AGP}(M, L, \lambda, t)$ unless $\mathrm{P} = \mathrm{NP}$, even for restricted L and M.

Theorem 6. *Unless* $\mathrm{P}=\mathrm{NP}$*, there does not exist any PTIME* n^l*-approximation algorithm for* $\mathsf{AGP}(M, L, \lambda, t)$*, even when* M *has a pipelined structure and when* L *is fixed.*

Proof. We show that unless $\mathrm{P}=\mathrm{NP}$, $\mathsf{AGP}_{\mathsf{max}}(M, L, \lambda, t)$ and $\mathsf{AGP}_{\mathsf{min}}(M, L, \lambda, t)$ do not allow any PTIME approximation algorithms with a polynomial approximation ratio.

(1) $\mathsf{AGP}_{\mathsf{max}}(M, L, \lambda, t)$. The main idea is by reduction from 3SAT: given an instance φ of 3SAT, we construct in PTIME a pipelined SWM M, a library L of fixed services, a realization constraint λ and an initial artifact t, such that (a) if φ is satisfiable, then there exists a realization ρ such that $M[\rho](t)=2^{|X|+1}$, where X is the set of variables in φ ($2^{|X|+1}$ is expressed in binary, in $O(|X|)$ space); and (b) otherwise for all realizations ρ, $M[\rho](t)=0$.

We next give the reduction. The mediator M, library L, constraint λ and initial artifact t are the same as their counterparts given in the proof of Theorem 5, except the following. The available service f_1 in L takes an artifact t as input, and converts the binary number $t.X$ into binary number Y such that Y and X have the same number of digits, and Y consists of 1 only. In addition, it sets $t.\mathsf{val}=Y$. Obviously (a) $M[\rho](t)$ is either 0 or $2^{|X|+1}$, and (b) the service function f_1 is in PTIME.

We now show the construction above is a reduction. Indeed, given any instance φ of 3SAT, (1) the mediator M, λ and t can be constructed in PTIME, and the algorithm $\mathcal{A}$ on M, L, λ and t is in PTIME, and (2) φ is satisfiable iff the algorithm returns a value no less than $\frac{1}{1+n^l}2^{|X|+1}$ for any given polynomial n^l. Assume by contradiction that there exists a PTIME n^l-approximation algorithm $\mathcal{A}$ for the aggregation problem, then one can decide 3SAT in PTIME. Hence, such an algorithm $\mathcal{A}$ cannot possibly exist unless $\mathrm{P}=\mathrm{NP}$.

Along the same lines as the proof of Theorem 5, it is straightforward to verify that if φ is satisfiable, then $M[\rho](t)=2^{|X|+1}$, and otherwise $M[\rho](t)=0$, as desired.

(2) $\mathsf{AGP}_{\mathsf{min}}(M, L, \lambda, t)$. The proof is similar, by reduction from non-tautology. Given an instance φ of the non-tautology problem, we construct in PTIME a pipelined SWM M, a realization constraint λ and an initial artifact t, with a fixed library L of available services, such that (a) if φ is not a tautology, then there exists a realization ρ such that $M[\rho](t)=2^{|X|+1}$, where X is the set of variables in φ, and (b) otherwise for all realizations ρ, $M[\rho](t)=0$.

This suffices. Assume that there exists a PTIME n^l-approximation algorithm $\mathcal{B}$ for the aggregation problem, then one can decide non-tautology in PTIME. Indeed, given any instance φ of non-tautology, we construct M, λ and t, executes $\mathcal{B}$ on M, L, λ and t, in PTIME; we can conclude that φ is a tautology iff the algorithm returns 0. This shows that unless $\mathrm{P}=\mathrm{NP}$, $\mathsf{AGP}_{\mathsf{min}}(M, L, \lambda, t)$ does not admit any PTIME n^l-approximation algorithm.

The reduction is the same as its counterpart given in the proof of Theorem 5, except that f_1 is changed as described in (1) above. □

5.2. DAG-structured mediators

We next investigate $\mathsf{AGP}(M, L, \lambda, t)$ for SWMs with a DAG structure. We show that like tree-structured SWMs, DAG-structured SWMs simplify the aggregation analysis: $\mathsf{AGP}(M, L, \lambda, t)$ is also decidable in this setting.

Given Theorem 5, the best one can hope for is that $\mathsf{AGP}(M, L, \lambda, t)$ remains in NP for DAG-structured SWMs. It turns out that for these SWMs, the complexity goes up, but the aggregation problem is still solvable in single-exponential time (in PSPACE). The PSPACE hardness bound remains intact even for fixed library L and deterministic realization constraint λ.

Theorem 7. $\mathsf{AGP}(M, L, \lambda, t)$ *is PSPACE-complete for DAG-structured SWMs. It remains PSPACE-hard when the library* L *is fixed and the realization constraint* λ *is deterministic.*

Proof. We show that $\mathsf{AGP}_{\mathsf{max}}(M, L, \lambda, t)$ and $\mathsf{AGP}_{\mathsf{min}}(M, L, \lambda, t)$ are in PSPACE, and are PSPACE-hard when L is fixed. We give a proof for $\mathsf{AGP}_{\mathsf{max}}(M, L, \lambda, t)$. The proof for $\mathsf{AGP}_{\mathsf{min}}(M, L, \lambda, t)$ is similar.

Upper bound. We first show that for any DAG-structured M, the problem is in PSPACE, by giving an NPSPACE algorithm. Given M, L, λ, t and a number B, the algorithm first guesses a realization ρ, and then checks whether ρ is valid *w.r.t.* λ and $M[\rho](t) \geqslant B$.

We show that the checking can be conducted in PSPACE. Indeed, checking whether ρ is valid is in PTIME. To check whether $M[\rho](t) \geqslant B$, the algorithm computes $M[\rho](t)$ as follows, constructing the execution tree ξ of the run in stages without storing the complete tree. At each node v of the tree labeled with (q, t, val), suppose that the transition and synthesis rules are $(q, \phi) \rightarrow (q_1, \tau_1), \ldots, (q_k, \tau_k)$ and $\mathsf{val}(q) \leftarrow F_q(\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k))$, respectively. The algorithm inspects the subtrees $\xi_1, \ldots, \xi_k$ of v one by one, for the successor states $q_1, \ldots, q_k$, respectively. To inspect a subtree, it follows a depth-first traversal order, and stores only necessary information. After a subtree ξ_j is checked, it retains only the val value of its root, denoted by val_j, and reuses its space to compute ξ_l for $l > j$. When all val_j's are available, $\mathsf{val}(v)$ is computed by evaluating $F_q(\mathsf{val}_1, \ldots, \mathsf{val}_j)$, and the space for storing $\mathsf{val}_1, \ldots, \mathsf{val}_k$ and the subtrees of v are released. The process starts from the root of ξ and proceeds until $M[\rho](t)$ is computed.

To see the space complexity of the checking, let d be the longest path in $G(M)$, w the maximum length of transition rules in M (the width of ξ), c_L the maximum space needed for evaluating an available service in L, and p_M the maximum space for evaluating a precondition or a synthesis function in M. Then at any stage of the computation, at most $O(w*d*|t|)$ space is needed to store necessary information for computing ξ, where w and d are linear in the size of M. In addition, at most $c_L + p_M$ space is required to evaluate preconditions and synthesis rules, where c_L and p_M are polynomials since all available services in L are PTIME functions, and all preconditions and synthesis rules are PTIME-computable. Putting these together, the algorithm is in $O(w*d*|t| + c_L + p_M)$ space. Since PSPACE = NPSPACE, the problem is in PSPACE.

Lower bound. We next show that the aggregation problem $\mathsf{AGP}_{\max}(M, L, \lambda, t)$ is PSPACE-hard when L is fixed, λ is deterministic and M has a DAG structure, by reduction from Q3SAT.

An instance of Q3SAT is given by a well-formed quantified Boolean sentence of the form $\varphi = Q_1 x_1 Q_2 x_2 \cdots Q_m x_m E$, where $E = C_1 \wedge \cdots \wedge C_n$ is an instance of 3SAT in which all the variables are $x_1, \ldots, x_m$, and $Q_i \in \{\forall, \exists\}$ for $i \in [1, m]$. The Q3SAT problem is to decide, given such a sentence φ, whether φ is valid. It is known that Q3SAT is PSPACE-complete (cf. [42]).

Given an instance φ of Q3SAT, we construct a DAG-structured SWM M, a fixed library L of available services, a realization constraint λ, an initial artifact t and a number B, such that φ is satisfiable iff there exists a realization ρ that is valid *w.r.t.* λ and makes $M[\rho](t) \geqslant B$.

(1) The artifact schema R_φ, initial artifact t, library L and number B are the same as their counterparts given in the proof of Theorem 5. Recall that L is fixed: it is independent of φ.

(2) The SWM M is defined as (Q, δ, σ, q_1), where $Q = \{q_j \mid j \in [1, m+2]\}$, and δ and σ are given as follows. For each $j \in [1, m]$, if the quantifier Q_j is $\forall$, then

$$(q_j, \mathsf{true}) \rightarrow (q_{j+1}, \tau_T), (q_{j+1}, \tau_F),$$
$$\mathsf{val}(q_j) \leftarrow \min\big(\mathsf{val}_1(q_{j+1}), \mathsf{val}_2(q_{j+1})\big),$$

where for $l \in [1, 2]$, $\mathsf{val}_l(q_{j+1})$ denotes the val value of the l-th successor state.

If the quantifier Q_j is $\exists$, then

$$(q_j, \mathsf{true}) \rightarrow (q_{j+1}, \tau_T), (q_{j+1}, \tau_F),$$
$$\mathsf{val}(q_j) \leftarrow \max\big(\mathsf{val}_1(q_{j+1}), \mathsf{val}_2(q_{j+1})\big).$$

For $j \in [m+1, m+2]$, we define

$$(q_{m+1}, \phi) \rightarrow (q_{m+2}, \tau_v), \ \mathsf{val}(q_{m+1}) \leftarrow \mathsf{val}(q_{m+2}),$$
$$(q_{m+2}, \mathsf{true}) \rightarrow . \qquad \mathsf{val}(q_{m+2}) \leftarrow .$$

The set $\Gamma(M)$ has three templates τ_T, τ_F, τ_v.

(3) The constraint λ is defined as follows: $\lambda(\tau_T) = \{f_T\}$, $\lambda(\tau_F) = \{f_F\}$, and $\lambda(\tau_v) = \{f_1\}$. Note that there is a unique ρ valid *w.r.t.* λ, i.e., λ is deterministic.

Intuitively, the SWM generates a complete binary tree of depth m to inspect all possible truth assignments for variables in φ. In other words, each path of length m encodes a truth assignment for φ. The end of the path is followed by a node labeled with state q_{m+1}, in which the precondition p evaluates the truth value of E based on the truth assignment $t.X$; if the condition is satisfied, $t.\mathsf{val}$ is changed to 1 by invoking f_1; otherwise $t.\mathsf{val}$ remains to be 0. The synthesis rules inspect whether φ is satisfied along all paths in the execution tree of a run, by using aggregation operators. More specifically, $\min$ is used to encode universally quantified variable x_j, and assures that both truth values for x_j are inspected. On the other hand, $\max$ encodes an existentially quantified variable x_j, and takes the truth value that satisfies φ if there exists any.

We show that the construction given above is indeed a reduction. Observe that there is a unique realization ρ valid *w.r.t.* λ. Assume that φ is true. It is easy to verify by induction on m that $M[\rho](t) = 1$, i.e., $M[\rho](t) \geqslant B$. Conversely, suppose that there exists a realization ρ such that $M[\rho](t) \geqslant 1$. One can verify that φ is true, again by a simple induction on m.

For $\mathsf{AGP}_{\min}(M, L, \lambda, t)$, the upper bound proof remains unchanged. The proof for the lower bound is the same except that B is now set to 0. Since a Q3SAT instance φ is a sentence with a unique truth value, φ is false iff there exists a realization ρ such that $M[\rho](t) \leqslant 0$. That is, the reduction given above also verifies that $\mathsf{AGP}_{\min}(M, L, \lambda, t)$ is PSPACE-hard when the library L is fixed and λ is deterministic. □

The case of the fixed mediator. We have seen from Theorems 2, 5, 6 and 7 that fixing library does not make our lives easier: the lower bounds remain unchanged when the library of available services is predefined and fixed.

Another practical setting is that a service provider often maintains a set of predefined mediators. That is, the SWMs can be considered fixed, while the library L is periodically updated by adding newly found available services to it, or removing obsolete services from it.

Below we show that fixing SWMs simplifies the aggregation synthesis: the problem is in PTIME for a fixed SWM M, when M has a tree or a DAG structure. Contrast this to Theorem 2, which tells us that when the mediators are recursively defined, fixing both mediators and library does not help.

Proposition 8. $\mathsf{AGP}(M, L, \lambda, t)$ *is in PTIME when M is a fixed DAG-structured SWM.*

Proof. Given L, λ, t and a realization ρ, it is in PTIME to compute $M[\rho](t)$ for a fixed DAG-structured SWM M. Indeed, it takes PTIME to construct the execution tree of the run of $M[\rho]$ on t, since the size of M is a constant when the SWM M is fixed.

In light of this, a PTIME algorithm for finding ρ that maximizes (or minimizes) $M[\rho](t)$ is as follows. For each realization ρ that is valid *w.r.t.* λ, computes $M[\rho](t)$, and returns the realization that yields the maximum (or minimum) output. The realizations can be enumerated by ranging over all $\lambda(\tau)$ for each template τ in M, where $|\Gamma(M)|$ is a constant when M is fixed. That is, there are at most $|L|^l$ many realizations, where $l = |\Gamma(M)|$ and it is a constant. The algorithm is obviously in PTIME. $\square$

6. Nondeterministic mediators

So far we have only considered deterministic mediators: in an SWM, each state is associated with a unique pair of transition rule and synthesis rule. As observed in, e.g., [44], in practice one may want to specify mediators with nondeterministic transition rules. For example, one may want to extend the SWM M_1 given in Example 2 by defining two pairs of transition and synthesis rules for state q_a with different preconditions, one for winter and the other for summer, such that different activities can be chosen in different seasons.

In this section we first extend SWMs and define nondeterministic SWMs (NSWMs), by allowing multiple pairs of transition and synthesis rules to be associated with each state. We then revisit the aggregation problem for NSWMs of various structures, showing that the presence of nondeterminism does not complicate the analysis. Finally, we formulate the termination problem, and establish its complexity bounds, for composite services generated from NSWMs and SWMs.

6.1. Nondeterministic SWMs

We first introduce nondeterministic SWMs.

Definition 6.1. A *nondeterministic synthesized mediator* (for web services; NSWM) over an artifact schema R_A is defined as $M = (Q, \delta, \sigma, q_0)$, where Q is a finite set of *states*, q_0 is the *start state*, δ is a set of *transition rules*, and σ is a set of *synthesis rules*, such that for each $q \in Q$, there exist a nonempty set of pairs $(\delta^i(q), \sigma^i(q))$ for $i \in [1, n_q]$, where $\delta^i(q)$ is a transition rule in δ and $\sigma^i(q)$ is a synthesis rule in σ, defined as follows:

$$\delta^i(q): \quad (q, \phi_i) \rightarrow (q_{i1}, \tau_{i1}), \ldots, (q_{ik}, \tau_{ik}),$$
$$\sigma^i(q): \quad \mathsf{val}(q) \leftarrow F_q\big(\mathsf{val}(q_{i1}), \ldots, \mathsf{val}(q_{ik})\big).$$

Here $q, q_1, \ldots, q_k$ refer to states in Q, τ_i's are template names from Γ and ϕ_i is a precondition, as given in Definition 2.1.

Obviously SWMs are a special case of NSWMs in which for each state q there exists a unique pair $(\delta(q), \sigma(q))$ of transition and synthesis rules, i.e., for each state q, $n_q = 1$.

A *realization* ρ for an NSWM M and the *composite service* $M[\rho]$ of M realized by ρ are defined the same as for an SWM; similarly for *realization constraints* λ (see Section 2).

On an input artifact t_0, a run of a composite service $M[\rho]$ of an NSWM M is, however, more involved than its counterpart for an SWM. In each state q of M, a pair $(\delta^i(q), \sigma^i(q))$ is now nondeterministically picked among n_q options, such that the transition and synthesis in this state follow the rules $\delta^i(q)$ and $\sigma^i(q)$, respectively. As a result, for the same t_0 there are (possibly exponentially many) different runs, which lead to different outputs. In other words, the results of different runs of $M[\rho]$ on t_0 no longer converge to yield the same output.

We revise the step relation $\Rightarrow_{(M[\rho], t_0)}$ for $M[\rho]$ of an NSWM M as follows. For two execution trees ξ and ξ', we write $\xi \Rightarrow_{(M[\rho], t_0)} \xi'$ if one of the following conditions holds.

Spawning. If there exists a leaf node v of ξ labeled with $(q, t, \perp)$, then there exists a pair $(\delta^i(q), \sigma^i(q))$ of transition and synthesis rules defined for state q, such that ξ' is obtained from ξ via the rule $\delta^i(q)$ as described in the spawning step of Section 2.2. That is, one of the rule pairs for q is nondeterministically chosen to expand ξ to ξ'.

Synthesizing. If there is no leaf node to which a transition rule applies, then ξ' is obtained from ξ by picking a node v labeled by $(q, t, \perp)$ so that none of its successors $u_1, \ldots, u_k$ has $\mathsf{val}(u_i) = \perp$, and updating $\mathsf{val}(v)$ according to the synthesis rule $\sigma^i(q)$ as described in the synthesizing step of Section 2.2. Here $\sigma^i(q)$ is the rule in the pair $(\delta^i(q), \sigma^i(q))$, where $\delta^i(q)$ is the transition rule chosen for spawning the children of v in the spawning phase.

If the process stops, the *result* of the run of $M[\rho]$ on artifact t_0 is an execution tree ξ such that $\xi_0 \Rightarrow^* \xi$ and there is no distinct ξ' such that $\xi \Rightarrow \xi'$. This yields the *output* of the run, which is the value of $\mathsf{val}(r)$ at the root r of ξ'.

If the process does not stop, we say that the result of the run is $\perp$, i.e., *undefined*.

Due to the nondeterministic nature of NSWM M, there are possibly multiple results of runs of $M[\rho]$ on the same artifact t_0 and hence, a set of outputs, denoted by $S(M[\rho], t_0)$.

When it comes to the aggregation analysis, we define $M[\rho](t_0)$ to be *the maximum value* (resp. *minimum*) in $S(M[\rho], t_0)$ for $\mathsf{AGP}_{\max}$ (resp. $\mathsf{AGP}_{\min}$), assuming that $\perp < m$ for any $m \in \mathbb{Q}$. Clearly $M[\rho](t_0)$ is well defined.

Example 4. We extend the SWM M_1 to an NSWM $M_2 = (Q_2, \delta_2, \sigma_2, q_2)$, where the set space Q_2 is $\{q_2, q_s, q_c, q_f, q_h, q_a^1, q_a^2, q_r, q_p, q_l\}$, and the transition rules δ_2 and synthesis rules σ_2 are the same as δ_1 and σ_1 shown in Fig. 2, except that for q_2 and q_a, rules are defined as follows:

$$
\begin{array}{ll}
(q_2, \mathsf{true}) \to (q_s, \tau_{id}) & \mathsf{val}(q_2) \leftarrow \mathsf{val}(q_s) \\
(q_2, \mathsf{true}) \to (q_c, \tau_{id}) & \mathsf{val}(q_2) \leftarrow \mathsf{val}(q_c) \\
\left(q_a, \phi_a^1\right) \to \left(q_a^1, \tau_a\right), (q_r, \tau_{id}) & \mathsf{val}(q_a) \leftarrow \mathsf{val}\left(q_a^1\right) + \mathsf{val}(q_r) \\
\left(q_a, \phi_a^2\right) \to \left(q_a^2, \tau_a\right), (q_r, \tau_{id}) & \mathsf{val}(q_a) \leftarrow \mathsf{val}\left(q_a^2\right) + \mathsf{val}(q_r) \\
\left(q_a^1, \mathsf{true}\right) \to (q_a, \tau_{id}) & \mathsf{val}\left(q_a^1\right) \leftarrow \mathsf{val}(q_a) \\
\left(q_a^2, \mathsf{true}\right) \to (q_a, \tau_{id}) & \mathsf{val}\left(q_a^2\right) \leftarrow \mathsf{val}(q_a)
\end{array}
$$

Here two pairs of rules are defined for the start state q_2, with the same precondition, while two pairs of rules are given for state q_a, with distinct preconditions for activities in the winter and the summer, respectively. In a run of $M_2[\rho]$ realized with ρ on an input artifact t_0, one pair of the rules for q_2 is nondeterministically picked, and $M_2[\rho](t_0)$ is the *minimum* among the outputs of all possible runs, for the minimization analysis. When the state q_a is reached, one pair of the rules for q_a is again nondeterministically chosen, as long as the precondition ϕ_a^1 (resp. ϕ_a^2) is satisfied.

6.2. The aggregation analysis of NSWMs

We now re-investigate the aggregation problem for NSWMs. For a predefined bound B and given an NSWM M over artifact schema R_A, an artifact t of R_A, a library L of available services and a realization constraint λ, *the aggregation problem* $\mathsf{AGP}(M, L, \lambda, t)$ is to determine whether there exists a realization ρ of M in L such that ρ is valid *w.r.t.* λ and $M[\rho](t) \geqslant B$ (for maximization $\mathsf{AGP}_{\max}$; resp. $M[\rho](t) \leqslant B$ for minimization $\mathsf{AGP}_{\min}$). In other words, it is to find realization ρ of M in L such that ρ is valid *w.r.t.* λ and there exists a run of $M[\rho]$ on t with the maximum (or minimum) output.

The main conclusion of this section is that NSWMs do not make the aggregation analysis harder. Indeed, the aggregation problem for NSWMs of various structures retain the same complexity as their SWM counterparts. To verify this, we first introduce a notion of normal forms for NSWMs and revise the notion of the mediator graphs for NSWMs.

A normal form of NSWMs. We show that every NSWM can be expressed in a "deterministic" form in which each state is associated with a unique transition rule and a unique synthesis rule. To do this we extend synthesis functions by allowing $F_q^n : \mathbb{Q}^k \cup \{\perp\} \to \mathbb{Q} \cup \{\perp\}$ such that $F_q^n(m_1, \ldots, m_k) = m_i$, where m_i is nondeterministically picked from $\{m_i \mid i \in [1, k], m_i \neq \perp\}$ if the set is nonempty, and $F_q^n(m_1, \ldots, m_k) = \perp$ if for all $i \in [1, k]$, $m_i = \perp$. We refer to F_q^n as a *choice selector*.

When choice selectors are used in the synthesis phase, they behave a little differently from synthesize functions described above. More specifically, for synthesizing a node v in an execution tree with F_q^n, $F_q^n(m_1, \ldots, m_k)$ is assigned to $\mathsf{val}(v)$ if there exists a child u_i of v such that $\mathsf{val}(u_i) \neq \perp$. That is, even when $\mathsf{val}(u_j) = \perp$ for $j \neq i$, i.e., when some children of v have an infinite subtree, $\mathsf{val}(v)$ can still be computed via F_q^n, and its value remains unchanged after a rational value is assigned to it.

Definition 6.2. We say that an NSWM $M = (Q, \delta, \sigma, q_0)$ is *in the normal form* if for each state $q \in Q$, there exist a unique transition rule $\delta(q)$ and a unique synthesis rule $\sigma(q)$ defined as follows:

$$\delta(q)\colon \quad (q, \phi) \to (q_1, \tau_1), \ldots, (q_k, \tau_k),$$

$$\sigma(q)\colon \quad \mathsf{val}(q) \leftarrow F_q\left(\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k)\right),$$

where F_q is either a synthesis function as in Definition 2.1 or a choice selector.

We say that two NSWMs M_1 and M_2 are *equivalent for aggregation* if for any input artifact t, library L and realization constraint λ, there exists a realization ρ_1 of M_1 in L valid *w.r.t.* λ iff there is a realization ρ_2 of M_2 in L valid *w.r.t.* λ such that $M_1[\rho_1](t) = M_2[\rho_2](t)$.

One can readily verify that each NSWM M can be "normalized" to an equivalent NSWM M', in PTIME. Hence the size $|M'|$ of M' is bounded by a polynomial in $|M|$.

Proposition 9. *For every NSWM M, an equivalent NSWM in the normal form can be computed in time polynomial in $|M|$.*

Proof. Given an NSWM $M = (Q, \delta, \sigma, q_0)$, we construct an NSWM $M' = (Q', \delta', \sigma', q_0)$ as follows. For each state $q \in Q$, suppose that q is associated with pairs $(\delta^i(q), \sigma^i(q))$ of transition and synthesis rules for $i \in [1, n_q]$, where

$$\begin{aligned} \delta^i(q): &\quad (q, \phi_i) \rightarrow (q_{i1}, \tau_{i1}), \ldots, (q_{ik}, \tau_{ik}), \\ \sigma^i(q): &\quad \mathsf{val}(q) \leftarrow F_q\big(\mathsf{val}(q_{i1}), \ldots, \mathsf{val}(q_{ik})\big). \end{aligned}$$

Then we introduce dummy states $q^1, \ldots, q^{n_q}$, and define a unique transition rule and a unique synthesis rule for q and q^i as follows:

$$\begin{aligned} \delta(q): &\quad (q, \mathsf{true}) \rightarrow \big(q^1, \tau_{id}\big), \ldots, \big(q^{n_q}, \tau_{id}\big), \\ \sigma(q): &\quad \mathsf{val}(q) \leftarrow F_q^n\big(\mathsf{val}\big(q^1\big), \ldots, \mathsf{val}\big(q^{n_1}\big)\big), \\ \delta\big(q^i\big): &\quad \big(q^i, \phi_i\big) \rightarrow (q_{i1}, \tau_{i1}), \ldots, (q_{ik}, \tau_{ik}), \\ \sigma\big(q^i\big): &\quad \mathsf{val}\big(q^i\big) \leftarrow F_q\big(\mathsf{val}(q_{i1}), \ldots, \mathsf{val}(q_{ik})\big). \end{aligned}$$

Here F_q^n is a choice selector, while F_q is a synthesis function as given in M.

Let Q' include all the states in Q as well as all dummy states introduced for each state $q \in Q$, and δ' and σ' be as defined as above. In a nutshell, M' enumerates all possible options of rules for each state, while nondeterministically returns one of the synthesized values for the state via a choice selector F_q^n. It is easy to verify that M' and M are equivalent for aggregation. Obviously M' is in the normal form and it can be computed in PTIME in $|M|$. $\square$

By Proposition 9, in the sequel we consider *w.l.o.g.* NSWMs in the normal form only.

The *mediator graph* $G(M)$ of an NSWM M can then be defined exactly the same as its counterpart for SWMs, as given in Section 3.

Complexity bounds. We are now ready to present the complexity bounds of the aggregation analysis of NSWMs with various underlying mediator graphs. We start with $\mathsf{AGP}(M, L, \lambda, t)$ when $G(M)$ is a cyclic graph. Since M is also an SWM, from Theorem 2 it follows immediately that the problem is undecidable even when M, L and λ are all fixed, and when $G(M)$ is a graph with a single self-loop.

Corollary 10. $\mathsf{AGP}(M, L, \lambda, t)$ *is undecidable for NSWMs even when NSWM M, library L and realization constraint λ are all fixed.*

When $G(M)$ is acyclic, $\mathsf{AGP}(M, L, \lambda, t)$ becomes decidable in single-exponential time. Indeed, compared to Theorems 5, 6 and 7, the result below tells us that NSWMs do not make our lives harder in this case.

Theorem 11. *For NSWMs, the problem* $\mathsf{AGP}(M, L, \lambda, t)$ *is*

1. *PSPACE-complete for DAG-structured M;*
2. *NP-complete for tree-structured M; and moreover,*
3. *there exists no PTIME n^l-approximation algorithm for any polynomial approximation ratio n^l even when M has a pipelined structure, unless $P = NP$.*

The lower bounds and approximation hardness remain unchanged even when library L is fixed.

Proof. The lower bounds in (1), (2) and (3) follow from Theorems 7, 5 and 6, respectively, since SWMs are a special case of NSWMs. Below we verify the upper bounds of (1) and (2). We give a proof for $\mathsf{AGP}_{\mathsf{max}}(M, L, \lambda, t)$. The proof for $\mathsf{AGP}_{\mathsf{min}}(M, L, \lambda, t)$ is similar.

(1) *DAG-structured NSWMs.* We first show that for any DAG-structured NSWM M, the problem is in PSPACE, by giving an NPSPACE algorithm. Given M, L, λ, t and a number B, the algorithm first guesses a realization ρ, and then checks whether (a) ρ is valid *w.r.t.* λ, and (b) there is a run of $M[\rho]$ such that the output of the run is no less than B. If so, then the algorithm returns "yes".

We show that the checking can be done in NPSPACE. By Proposition 9, we assume that M is in the normal form. (1) As shown in the proof of Theorem 7, it is in PTIME to check whether ρ is valid *w.r.t.* λ. (2) Moreover, computing the output of a run of $M[\rho]$ on t is in NPSPACE. (i) As in the proof of Theorem 7, the algorithm constructs the execution tree ξ of a run in stages without storing the complete tree. At each node v of the tree labeled with (q, t, val), the subtrees of v are inspected one by one. After a subtree ξ_i is checked, only the val value of its root is retained (denoted by val_i), and its space is reused for computing other subtrees. (ii) For synthesizing at node v, suppose that the synthesis rule for q is $\mathsf{val}(q) \leftarrow F_q(\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k))$. Consider the two cases of F_q: (a) when F_q is a synthesis function, $\mathsf{val}(v)$ is $F_q(\mathsf{val}_1, \ldots, \mathsf{val}_k)$; (b) when F_q is a choice selector, one of $\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k)$ is nondeterministically picked as the $\mathsf{val}(q)$ value at v. In both cases the space for storing $\mathsf{val}_1, \ldots, \mathsf{val}_k$ and the subtrees of v is released. The process proceeds until the output of the run is computed. As argued in the proof of Theorem 7, this can be done in NPSPACE. Since NPSPACE = PSPACE, the algorithm is in PSPACE.

(2) *Tree-structured NSWMs.* We now show that for tree-structured NSWMs M, the aggregation problem is in NP, by providing an NP algorithm. Given M, L, λ, t and a number B, the algorithm first guesses (a) a realization ρ, and (b) a run of $M[\rho]$ on t. It then checks whether the output of the run is no less than B. If so, $M[\rho](t) \geqslant B$ and the algorithm returns "yes".

By Proposition 9, we assume that M is in the normal form. Note that for a tree-structured NSWM M, the mediator graph $G(M)$ and the execution tree of any run of $M[\rho]$ are isomorphic, for any realization ρ. Hence we guess a run of $M[\rho]$ as follows: for any node v in an execution tree labeled with (q, t, val), suppose that the synthesis rule for q is $\mathsf{val}(q) \leftarrow F_q(\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k))$. If F_q is a choice selector, we nondeterministically pick $\mathsf{val}(q_i)$ as the value of $\mathsf{val}(v)$ for $i \in [1, q_k]$. This is independent of what realization ρ is chosen.

As shown in the proof of Theorem 5, it is in PTIME to check whether ρ is valid *w.r.t.* λ, and to compute the output of the run of $M[\rho]$ on t when M is tree structured. Thus, the algorithm is indeed in NP, and so is the aggregation analysis of tree-structured NSWMs. □

We have seen from Proposition 8 that for an SWM M, if M is fixed and $G(M)$ is acyclic, $\mathsf{AGP}(M, L, \lambda, t)$ becomes tractable. We show that it is also the case for NSWMs.

Proposition 12. *For a fixed DAG-structured NSWM M, the problem* $\mathsf{AGP}(M, L, \lambda, t)$ *is solvable in polynomial time.*

Proof. As shown in Proposition 8, when M is fixed, (a) there exist polynomially many realizations ρ of M in L that are valid *w.r.t.* λ, and (b) for each run of $M[\rho]$ on t, its execution tree can be constructed in PTIME. In addition, we show (c) that for each realization ρ, the number of runs of $M[\rho]$ on t is a constant. Indeed, by Proposition 9, we assume that M is in the normal form. As argued in the proof of Theorem 11(2), the execution tree of each run of $M[\rho]$ is isomorphic to the tree obtained by unfolding $G(M)$. Then a run of $M[\rho]$ is determined as follows. For any node v in an execution tree labeled with (q, t, val), suppose that the synthesis rule for q is $\mathsf{val}(q) \leftarrow F_q(\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k))$. If F_q is a choice selector, the run is decided by which $\mathsf{val}(q_i)$ is chosen as the value of $\mathsf{val}(v)$, for $i \in [1, q_k]$. When M is fixed, the size of M is a constant; hence so are the number of runs of $M[\rho]$ for a realization ρ, and the size of the execution tree of each run.

This yields a PTIME algorithm for finding ρ such that $M[\rho](t)$ is maximum (or minimum). The algorithms ranges over all realizations ρ that are valid *w.r.t.* λ, and for each such ρ, it ranges over all runs of $M[\rho]$ to compute the output of the run. It returns the realization ρ and the run that maximizes (or minimizes) $M[\rho](t)$. From the discussion above it follows that the algorithm is indeed in PTIME. □

6.3. Termination analysis

We have seen that a composite service $M[\rho]$ generated from an NSWM or an SWM may not terminate on an input artifact t, i.e., there exists no any finite execution tree that is the result of a run of $M[\rho]$ on t. In other words, $M[\rho](t)$ is not defined. Hence it is natural to investigate the *termination* problem, denoted as TMP and presented as follows.

PROBLEM:	$\mathsf{TMP}(M, L, \lambda, t)$
INPUT:	1. an NSWM M on artifact schema R_A; 2. an artifact t of R_A; 3. a library L of available services; 4. a realization constraint λ.
QUESTION:	Does there exist a realization ρ of M in L such that ρ is valid *w.r.t.* λ and $M[\rho](t)$ is defined, i.e., a run of $M[\rho]$ terminates?

The termination analysis is nontrivial for NSWM M when $G(M)$ is a cyclic graph. In contrast, it is trivial when $G(M)$ is acyclic.

Corollary 13. *For an NSWM M with mediator graph $G(M)$, the problem* $\mathsf{TMP}(M, L, \lambda, t)$ *is*

1. *undecidable if $G(M)$ is cyclic, even when M is a fixed SWM, and when library L and realization constraint λ are fixed*; *and*
2. *in linear-time if $G(M)$ is a DAG or a tree.*

Proof. (1) We show that $\mathsf{TMP}(M, L, \lambda, t)$ is undecidable by reduction from the problem of determining solutions of Diophantine equations with fixed-degree and fixed number of variables. Given a Diophantine equation p, the reduction is the same as the one given in the proof of Theorem 2, in which $G(M_0)$ is a self-loop, M_0 is a fixed SWM, and both λ and L are fixed. As shown there, p has a solution in $\mathbb{N}^k$ if and only if $M_0[\rho]$ terminates and returns 0.

(2) When $G(M)$ is acyclic, one can verify that for any realization ρ valid *w.r.t.* λ, every run of $M[\rho]$ on an artifact t always terminates. Hence the termination problem in this case is equivalent to the problem for checking whether there exists a realization valid *w.r.t.* λ, which takes linear time in the size of $\Gamma(M)$. $\square$

7. Conclusion

We have given a formal treatment of the aggregation synthesis of Web service mediators, a problem that is of practical importance but has not been adequately addressed theoretically. We have developed a model for specifying mediators with aggregation synthesis, deterministic or nondeterministic, and formulated the aggregation problem. We have also established matching upper and lower bounds on the problem for mediators of various structures. In addition, we have provided complexity bounds of the termination analysis of composite services of various structures generated from mediators.

The main results of the paper are summarized in Table 1 (Section 1). We have shown that the problem is beyond reach in practice for recursively defined mediators, even when the mediators and the library of available services are predefined and fixed. Nevertheless, for mediators with a DAG or a tree structure, the problem becomes decidable in single-exponential time, which is an acceptable complexity for static analysis problems. More specifically, the problem is PSPACE-complete for DAG-shaped mediators, and is NP-complete for tree-shaped ones; it is even in PTIME when a set of predefined mediators are considered, a common setting in practice. These make our lives easier, but only to some extent: the NP-lower bound remains intact when the mediator has a pipelined structure and the library is fixed. Worse still, the problem does not allow any PTIME approximation algorithms with a polynomial ratio. The PSPACE lower bound is also robust: it remains unchanged when the library is fixed.

This work is a first step toward understanding the aggregation synthesis of Web services. There is naturally much more to be done. First, a run of a composite service generated from an NSWM may not terminate. We are currently investigating practical restrictions on NSWMs such that every run is guaranteed to terminate and yield a solution. Second, while the aggregation problem is undecidable in general and is intractable for non-recursive NSWMs, we expect that practical PTIME cases can be identified in certain specific application domains. Third, it is interesting to revisit the composition problem when aggregation synthesis is brought into the play. That is, we want to automatically generate mediators that coordinate available services and deliver a requested service, with aggregation analysis that aims to best serve the users' need. Finally, we would like to develop efficient heuristic algorithms to realize mediators with aggregation synthesis for specific applications. The operational semantics given in Section 2.2 provides a conceptual-level strategy for delivering a requested service. The strategy can certainly be improved by capitalizing on practical pruning techniques. For instance, one may employ a lazy evaluation strategy such that if some branch already yields a larger (or smaller) value than a given bound, realizations of the templates in other branches as well as their computation can be entirely avoided.

Acknowledgments

Fan is supported in part by an IBM scalable data analytics for a smarter planet innovation award, the RSE-NSFC Joint Project Scheme, EPSRC EP/J015377/1, as well as the 973 Program 2012CB316200 and NSFC 61133002 of China. Deng is supported in part by 863 2012AA011203, NSFC 61103031 and CPSF 2011M500218 of China. Libkin is supported by EPSRC grants EP/G049165/1 and EP/J015377/1, and FET-Open Project FoX, grant agreement 233599.

References

[1] D. Berardi, D. Calvanese, G.D. Giacomo, M. Lenzerini, M. Mecella, Automatic service composition based on behavioral descriptions, Int. J. Coop. Inf. Syst. 14 (4) (2005) 333–376.
[2] X. Fu, T. Bultan, J. Su, Analysis of interacting BPEL Web services, in: Proc. 12th Int. World Wide Web Conf., 2004, pp. 621–630.
[3] C.E. Gerede, R. Hull, O.H. Ibarra, J. Su, Automated composition of e-services: lookaheads, in: Proc. 2nd Int. Conf., Service-Oriented Computing, 2004, pp. 252–262.
[4] S. Abiteboul, V. Vianu, B.S. Fordham, Y. Yesha, Relational transducers for electronic commerce, J. Comput. System Sci. 61 (2) (2000) 236–269.
[5] D. Berardi, D. Calvanese, G.D. Giacomo, R. Hull, M. Mecella, Automatic composition of transition-based semantic web services with messaging, in: Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 613–624.
[6] A. Deutsch, L. Sui, V. Vianu, Specification and verification of data-driven Web applications, J. Comput. System Sci. 73 (3) (2007) 442–474.
[7] A. Deutsch, L. Sui, V. Vianu, D. Zhou, Verification of communicating data-driven Web services, in: Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 90–99.
[8] W. Fan, F. Geerts, W. Gelade, F. Neven, A. Poggi, Complexity and composition of synthesized Web services, in: Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2008, pp. 231–240.
[9] M. Spielmann, Verification of relational transducers for electronic commerce, J. Comput. System Sci. 66 (1) (2003) 40–65.
[10] K. Bhattacharya, C.E. Gerede, R. Hull, R. Liu, J. Su, Towards formal analysis of artifact-centric business process models, in: Proc. 5th Int. Conf. Business Process Management (BPM), 2007, pp. 288–304.
[11] A. Deutsch, R. Hull, F. Patrizi, V. Vianu, Automatic verification of data-centric business processes, in: Proc. 12th Int. Conf. on Database Theory, 2009, pp. 252–267.

[12] C. Fritz, R. Hull, J. Su, Automatic construction of simple artifact-based business processes, in: Proc. 12th Int. Conf. on Database Theory, 2009, pp. 225–238.
[13] R. Hull, Artifact-centric business process models: brief survey of research results and challenges, in: Proc. OTM Confederated Int. Conf. On the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS, Volume Part II (OTM Conferences (2)), 2008, pp. 1152–1163.
[14] S. Abiteboul, L. Segoufin, V. Vianu, Static analysis of active XML systems, in: Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2008, pp. 221–230.
[15] X. Fu, T. Bultan, J. Su, Conversation protocols: a formalism for specification and verification of reactive electronic services, Theoret. Comput. Sci. 328 (1–2) (2004) 19–37.
[16] E. Damaggio, A. Deutsch, V. Vianu, Artifact systems with data dependencies and arithmetic, in: Proc. 14th Int. Conf. on Database Theory, 2011, pp. 66–77.
[17] Y. Lustig, M.Y. Vardi, Synthesis from component libraries, in: Proc. 12th Int. Conf. Foundations of Software Science and Computational Structures (FoSSaCS), 2009, pp. 395–409.
[18] A. Muscholl, I. Walukiewicz, A lower bound on Web services composition, in: Proceedings of International Conference on Foundations of Software Science and Computational Structures (FoSSaCS), 2007, pp. 274–286.
[19] Y. Zhang, W. Fan, et al. Extending online travel agency with adaptive reservations, in: Proc. OTM Confederated Int. Conf. on the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS, Volume Part I (OTM Conference (1)), 2007, pp. 285–299.
[20] IBM, The Océano project, http://researchweb.watson.ibm.com/oceanoproject/, 2002.
[21] A. Nigram, N. Caswell, Business artifacts: an approach to operational specification, IBM Syst. J. 42 (3) (2003) 428–445.
[22] S. Nakajima, Verification of web service flows with model-checking techniques, in: Proc. 1st Int. Symp. on Cyber Worlds (CW), 2002, pp. 78–385.
[23] E. Clarke, O. Grumberg, D. Peled, Model Checking, The MIT Press, 1999.
[24] T. Deng, W. Fan, L. Libkin, Y. Wu, On the aggregation problem for synthesized web services, in: Proc. 13th Int. Conf. on Database Theory, 2010, pp. 242–251.
[25] P.A. Bonatti, P. Festa, On optimal service selection, in: Proc. 14th Int. World Wide Web Conf., 2005, pp. 530–538.
[26] S.-Y. Hwang, H. Wang, J. Tang, J. Srivastava, A probabilistic approach to modeling and estimating the QoS of web-services-based workflows, Inform. Sci. 177 (23) (2007) 5484–5503.
[27] T. Yu, K.-J. Lin, Service selection algorithms for Web services with end-to-end QoS constraints, Inf. Syst. E-Bus. Manag. 3 (2) (2005) 103–126.
[28] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, H. Chang, QoS-aware middleware for Web services composition, IEEE Trans. Softw. Eng. 30 (5) (2004) 311–327.
[29] A. Simitsis, K. Wilkinson, M. Castellanos, U. Dayal, Qox-driven etl design: reducing the cost of etl consulting engagements, in: SIGMOD, 2009, pp. 953–960.
[30] D. Deutch, T. Milo, N. Polyzotis, T. Yam, Optimal top-k query evaluation for weighted business processes, PVLDB 3 (2010) 940–951.
[31] D. Deutch, T. Milo, Evaluating top-k queries over business processes, in: Proc. 25th Int. Conf. on Data Engineering, 2009, pp. 1195–1198.
[32] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl, 2001.
[33] Web Services Conversation Language (WSCL) 1.0, http://www.w3.org/TR/wscl10/, 2002.
[34] OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/, 2004.
[35] Semantic Web Services Framework, http://www.daml.org/services/swsf/1.1/, 2005.
[36] Business Process Execution Language for Web Services version 1.1 (BEPL4WS), http://www.ibm.com/developerworks/library/specification/ws-bpel/, 2004.
[37] C. Schmidt, M. Parashar, A peer-to-peer approach to web service discovery, World Wide Web J. 7 (2) (2004) 211–229.
[38] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, F. Toumani, On automating Web services discovery, VLDB J. 14 (1) (2004) 84–96.
[39] M. Pistore, P. Roberti, P. Traverso, Process-level composition of executable web services: "on-the-fly" versus "once-for-all" composition, in: Proc. European Conf. on the Semantic Web: Research and Applications (ESWC), 2005, pp. 62–77.
[40] Y. Matijasevic, Some purely mathematical results inspired by mathematical logic, in: Logic, Foundations of Mathematics and Computability Theory, in: Western Ontario Series in Philosophy of Science, vol. 9, Reidel, 1975, pp. 121–127.
[41] J.P. Jones, Undecidable Diophantine equations, Bull. Amer. Math. Soc. 3 (2) (1980) 859–862.
[42] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, 1994.
[43] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer-Verlag, 1999.
[44] D. Berardi, G.D. Giacomo, M. Mecella, D. Calvanese, Composing web services with nondeterministic behavior, in: Proc. IEEE Int. Conf. on Web Services, 2006, pp. 909–912.