Automatic Construction of Simple Artifact-based Business Processes

Christian Fritz*
Dept. of Computer Science
University of Toronto
fritz@cs.toronto.edu

Richard Hull†
Watson Research Center
IBM
hull@us.ibm.com

Jianwen Su
Dept. of Computer Science
U C Santa Barbara
su@cs.ucsb.edu

ABSTRACT

Almost all medium- and large-scale businesses rely on electronic workflow systems to manage their business processes. A key challenge is to enable the easy re-use and modification of these workflow schemas and their piece-parts, so that they can be adapted to new business situations. This paper describes an approach for automatic construction (and thus, evolution) of a workflow schema that satisfies a specified condition (or "goal"), starting from a set of basic building block services (or "tasks"). We use a workflow model based on "business artifacts", which represent key (real or conceptual) business entities, and include both the business-relevant data about them and a specification of their lifecycle, that is, how they can evolve over time as they move through the workflow as the result of services being applied to them.

This paper uses a declarative form of artifact-centric workflow. The services are non-deterministic, which corresponds to the intuition that humans performing the services may rely on information that is not modeled within the framework. We study the problem of, given a goal to be achieved, automatically finding the "maximal" workflow schema that has the following property: every execution is either complete or can be completed, and every complete execution satisfies the goal. We also study a complimentary problem, in which exception-handling is used to deal with executions that would otherwise not complete successfully. These problems are non-trivial because the workflow services are non-deterministic.

This paper provides a general framework for studying these problems, and shows a tight relationship between workflow systems specified using logics that permit quantifier elimination and the ability to construct maximal schemas with the desired properties. The paper then studies a restricted setting to provide insights into complexity issues. Even in the restricted setting, the problem of testing properties of maximal workflows is PSPACE-complete.

*Part of work by this author performed while visiting Bell Laboratories, Alcatel-Lucent Technologies.

†Part of work by this author performed while employed by Bell Laboratories, Alcatel-Lucent Technologies.

ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.

1. INTRODUCTION

Business process management refers to the general challenge of enabling the flexible yet systematic management and control of the operations of a business (or similar organization). Almost invariably, electronic workflow systems are used to support the management of business processes in medium- and large-sized organizations. There has been a rapid increase in demand for the construction, execution, and management of workflows in which data are created and managed in a coherent manner that cuts across large portions of the workflow schema. A key contributing factor is the availability of mature data management tools which make it very easy to create, store, and access numerous data sources. Another is the growing popularity of the Service-Oriented Architecture (SOA), a methodology to (re)organize complex software systems in terms of piece-wise sharable, individually manageable "services" that are easily accessible from a network. The availability of data sources and services has been enabling practitioners in business enterprises to build application systems of ever growing size and complexity. However, a serious challenge facing the practitioners is the lack of fundamental principles, techniques, and tools necessary to deal with issues in managing assemblies of large set of data sources and services. This paper is among initial efforts [6, 14, 19, 18] towards discovering design principles for business processes through investigation of fundamental issues in workflow management.

A critical challenge in business process management is to enable the easy re-use and modification of workflow schemas and their piece-parts, so that they can be adapted to new business situations. Earlier and current work in SOA and workflow areas emphasize the process assembly aspect, and treat the data exchange and maintenance as either an afterthought or a consideration local to individual services (see discussions in [6, 8]). Experiences in workflow management have revealed that serious problems in assembling workflows from existing services often arise from the lack of coherent modeling of the data at a level that is global to the entire workflow, and several efforts have worked to give data more prominence, e.g., [21, 26, 43]. IBM Research has been addressing this area by introducing and extending the "artifact-centric" approach to workflow and business process management. A watershed paper is [37], and there has been considerable research and development based on that work, e.g., [5, 6, 7, 8, 24, 31, 32, 41]. The artifact-centric approach enables business managers to have better insight into their business operations, and has been shown to substantially reduce the cost of business transformations [5]. Artifacts represent key (real or conceptual) business entities, and include both the business-relevant data about them and a specification of their lifecycle, that is, how they can evolve over time as they move through the workflow as

the result of services being applied to them. Unlike process-centric models, the artifact approach puts data squarely in the center stage and models a workflow by adding services that act on the data. In a sense, this approach beautifully bridges the significant gap between the semantic web services work in academic research communities and the usage of web services and SOA in business development contexts (a challenge raised in a panel at ICWS '04).

In spite of the initial success of applying artifact modeling methodology in business process management [7, 5, 41], there is a lack of fundamental understanding of the approach with respect to re-use and modification of workflows. In [6, 18, 19], we developed formal models for artifact-based workflow and studied static analysis of several properties such as satisfiability, "dead-end", reachability, etc. Additional research on static analysis is reported in [14]. In this paper, we make our first attempt at automating workflow construction, that is, workflow synthesis. We focus on issues including (1) whether it even makes sense to attempt to construct a workflow for a given goal, (2) construction of a workflow that permits all input artifacts which are guaranteed to support successful executions, (3) goal-directed workflow construction that permit exceptions. The workflows constructed can give insights, at design time, about the properties of a goal, which can help the goal designer to understand whether the goal is capturing her intended semantics.

This paper uses a declarative form of artifact-centric workflow, that follows and significantly extends the model of [6]. We study the problem of, given a goal to be achieved, finding the "maximal" workflow schema that has the following property: every execution is either complete or can be completed, and every complete execution satisfies the goal. This problem is complicated by the fact that often the effect of running a service against an artifact is not completely known during workflow construction. Thus we model these effects as non-deterministic and construct workflows conservatively, i.e. such that they succeed to achieve the goal for all possible outcomes of the involved service. We also study a complimentary problem, in which exception-handling is used to deal with executions that would otherwise not complete successfully. The formulations expose some intricate subtleties perhaps for the first time in the formal setting.

This paper provides a general framework for studying these problems, and shows a tight relationship between workflow systems specified using logics that permit quantifier elimination and the ability to construct maximal schemas with the desired properties. The paper then studies a restricted setting to provide insights into complexity issues. Even in our restricted setting, the problem of finding maximal workflows is PSPACE-complete. The paper identifies a key property of workflows that can contribute to increased run-time costs in these constructions.

Although the main technical results reported here are in a restricted setting, we view them as an important starting point for further research into mechanisms for both fully- and semi-automated construction of workflows from goal specifications. In general terms, a formal framework and algorithms for automatic generation of workflow schemas from goal specifications holds the promise of dramatic simplification of both workflow design and workflow evolution. In the case of design, automatic generation of the maximal workflow schema associated with a goal will allow designers and managers to focus largely on their business goals, rather than getting bogged down with procedural realizations of those goals. In the case of evolution, there is the possibility that business managers can adjust the goal specification for their workflow, and te workflow schema can be adjusted automatically to a corresponding schema.

Artifact Class: `PurchaseOrder`
Associated Attributes:
  `prodName`: string
  `prodType`: {"hw", "sw"}
  `bid`: integer
  `profitMargin`: [0..100] // as a percentage
  `approved`: bool
  `execApproved`: bool
  `scheduleDate`: date
  `archived`: bool

Figure 1: A simple artifact class `PurchaseOrder`

This paper is organized as follows. Section 2 provides a motivating example to illustrate both the workflow model and the focused technical problems. Section 3 establishes the formal model. Section 4 places our problem space into a generalized logic setting, and provides a close linkage between quantifier elimination and the ability to construct maximal workflows that use quantifier-free conditions and rules. Section 5 introduces and studies the restricted setting. Section 6 briefly discusses some related work from various areas of Computer Science. Section 7 provides brief conclusions and selected research directions.

2. MOTIVATING EXAMPLE

This section provides an informal introduction to the artifact-based workflow framework, presents a motivating example, and illustrates the main technical results that the current paper is focused on. The formal definitions and statements of results appear in the following sections.

Workflows in the artifact-based approach are focused on *artifacts*, which are essentially sets of attribute/value pairs. Most values start out undefined, and become defined during the life-cycle of the artifact. (In the general artifact-based model [37, 6, 19], artifacts can also move, during their life-cycle, through the states of a finite state machine. We do not consider artifact states in this paper.)

An example artifact *class*, called `PurchaseOrder`, along with an associated set of attributes, is shown in Figure 1. In our examples, we shall assume that `prodName`, `prodType`, and `bid` are initialized. Attribute `bid` is intended to hold the value that is offered by a potential buyer for the product `prodName`.

Artifacts are manipulated by services, as illustrated next.

EXAMPLE 2.1. A representative family of services that operate on artifacts of `PurchaseOrder` is shown in Figure 2. The `Estimate` service computes the `profitMargin` of a purchase request, based on the `prodName`, the `prodType` and the `bid`. This service can be executed on an artifact o only if the *pre-condition* PRE is satisfied by o. The *conditional effect* EFF describes properties of the outcome of executing the service (Note that $true \rightarrow \phi$ denotes the unconditional effect ϕ). Although the complete algorithm of `Estimate` is not known, according to EFF we do know that if the `bid` is $\leqslant 400$ then the `profitMargin` will end up $\leqslant$ 25%, and if the `bid` is > 350 then the `profitMargin` will end up $>$ 20%.

Estimate: profitMargin
  PRE: DEF(prodName) ∧ DEF(prodType) ∧ DEF(bid)
  EFF:
    bid ⩽ 400 → profitMargin ⩽ 25%
    bid > 350 → profitMargin > 20%
RoutineApproval: approved
  PRE: DEF(bid) ∧ DEF(profitMargin)
  EFF:
    bid ⩽ 100 → approved = *true*
    prodType = "sw" → approved = *true*
    prodType = "hw" ∧ profitMargin > 10%
                → approved = *true*
    "else" → approved = *false*
ExecApproval: execApproved
  PRE: approved = *false*
  EFF: *true* → DEF(execApproved)
Schedule: scheduleDate
  PRE: DEF(prodName)
  EFF: true→ DEF(scheduleDate)
Archive: archived
  PRE: DEF(scheduleDate) ∨ execApprov = *false*
  EFF: *true* → archived = *true*

Figure 2: Family of services associated with PurchaseOrder artifact class

Service RoutineApproval also has a rich conditional effect. We have used a shorthand for the fourth formula of EFF: here the "else" means to take the negation of the disjunction of the conditions of the first three conditional effects. Service RoutineApproval will always define a value for approved, perhaps true (i.e., approved) and perhaps false (i.e., not approved).

Service ExecApproval can be invoked if RoutineApproval ended with false. Intuitively, this service gives an executive a chance to override a non-approval that might occur from RoutingApproval. This service will always define a value for execApproved, even if the executive denies to approval.

Schedule is performed based only on knowledge of the prodName; intuitively this schedules a date for the creation or delivery of the product.

Intuitively, Archive will be used to represent the closing of a workflow execution. This can be achieved only if a date has been scheduled or if execApproved is defined and false. ∎

Speaking informally, a *pre-workflow* (*schema*) is a pair $(\mathcal{A}, \mathcal{S})$, where $\mathcal{A}$ is a family of artifact classes with associated attribute sets, and $\mathcal{S}$ is a family of services which can act on artifacts in these classes. The artifact class PurchaseOrder, along with the set of services in Figure 2, forms the pre-workflow called here Purchasing.

In the general artifact model, multiple artifacts may interact with each other. In this section and Section 5 we focus on a single artifact class, and assume that the individual artifacts of this class evolve independently of each other.

Given a pre-workflow $\mathcal{P}$, a path over $\mathcal{P}$ is modeled as a sequence $\vec{s} = s_1, \ldots, s_n$ of "snapshot". Each snapshot corresponds to a state of the overall system, and consists of a set of artifacts along within information about the values of the attributes which are defined for each artifact. Also, s_{i+1} corresponds to the application of one service from $\mathcal{P}$ to one artifact occurring in s_i. (In this paper we do not consider artifact "creation".) We focus in this paper on how a single artifact o can evolve from some initial state through other states as the result of service invocations. Thus we focus on path $\vec{o} = o_1, ..., o_n$ for a single artifact o, and for each i, the artifact (instance) o_{i+1} corresponds to the result of applying some service to o_i.

The artifact-based approach to workflow permits a declarative style of workflow construction called here *goal-based workflow construction*. In particular, suppose that we are given a pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$. We also assume that we are given a set $\mathcal{I}$ of attributes that are to be initialized for all input artifacts. We are interested in constructing an actual workflow $\mathcal{W}$ from $\mathcal{P}$ such that all executions in $\mathcal{W}$ satisfy a goal γ. In this paper we consider goals γ which are logical formulas to be satisfied by the final snapshot of an execution. (More generally, the goal γ might include temporal properties to be satisfied by the overall execution.)

The following is a very natural, first question that arises when contemplating goal-driven workflow construction.

Q1 (Satisfiability): Given pre-workflow $\mathcal{P}$ (with fixed set of input attributes to be initialized) and goal γ is there an initial artifact o and a path on o in $\mathcal{P}$ which ends in a snapshot that satisfies γ?

EXAMPLE 2.2. In connection with the PurchaseOrder class and the services of Example 2.1, consider the goal

$$\begin{aligned} \gamma_1 \quad = \quad & \texttt{archived} = \mathit{true} && c1 \\ & \wedge (\texttt{DEF}(\texttt{scheduleDate}) \\ & \quad \rightarrow (\texttt{approved} = \mathit{true} && c2 \\ & \quad\quad \vee \texttt{execApproved} = \mathit{true})) \end{aligned}$$

Intuitively, clause $c1$ here states that a path is successful for γ_1 only if it includes execution of the Archive service. Clause $c2$ (which consists of the second, third, and fourth lines of the formula) states that a purchase can actually be scheduled only if it is approved by RoutineApproval or by ExecApproval.

In connection with **Q1**, it is easy to construct artifacts which have successful paths through this pre-workflow. For example, an input artifact with prodType = "sw" can progress through Estimate, then RoutineApproval, then Schedule, and finally pass through Archive. ∎

Given a pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$, we create a workflow by associating a set $\mathcal{R}$ of (transition) *rules* to $\mathcal{P}$. The problem of goal-directed workflow construction thus becomes the problem of finding a ruleset that completes a pre-workflow and complies with the goal.

EXAMPLE 2.3. Recall goal γ_1 from the preceding example. Figure 3 provides a diagrammatic representation of a set of rules for Purchasing. Two of the rules illustrated there are

if DEF(prodName) ∧ DEF(prodType) ∧ DEF(*bid*)
  invoke Estimate
if (approved = *true* ∨ execApproved = *true*)
  invoke Schedule

Speaking intuitively, the ruleset $\mathcal{R}_1$ indicated in Figure 3, along with various properties of the services themselves, will guide each

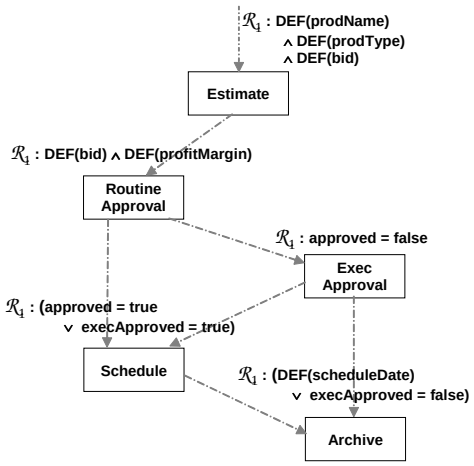

Figure 3: Simple ruleset that is γ_1-safe

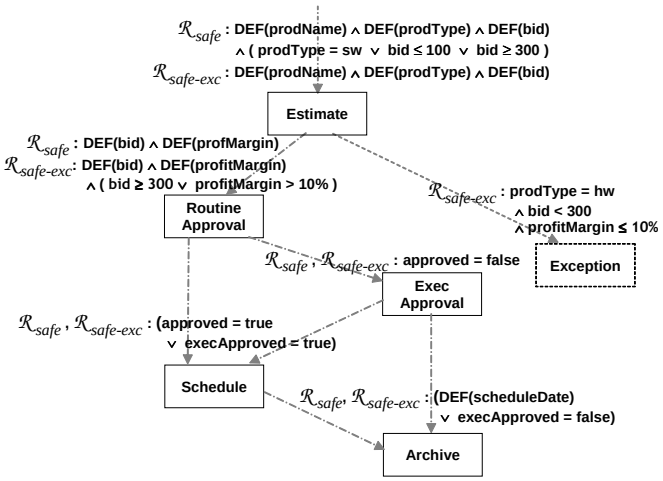

Figure 4: Alternative rendition of Maximal workflows for Purchasing pre-workflow and goal γ_2

execution through a certain sequencing of services, and will guarantee that each non-extendable execution satisfies γ_1. ∎

Given a pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ and a ruleset $\mathcal{R}$ for $\mathcal{P}$, we say that $\mathcal{W} = (\mathcal{A}, \mathcal{S}, \mathcal{R})$ is a workflow. In general, multiple rules may call for invoking the same service. The rules indicate permissions to invoke a service. However, the truth of some rule condition does not require that the service actually be invoked.

The workflow $\mathcal{W} = (\mathcal{A}, \mathcal{P}, \mathcal{R})$ is *safe* for goal γ if each (non-extendable) execution for $\mathcal{W}$ ends in a snapshot that satisfies γ. The pre-workflow `Purchasing` extended by $\mathcal{R}_1$ of Example 2.3 is γ_1-safe.

It is now natural to ask the following.

Q2 (maximal ruleset γ-safe for $\mathcal{P}$): Given pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ and goal γ, is there a ruleset $\mathcal{R}$ such that $(\mathcal{A}, \mathcal{S}, \mathcal{R})$ is γ-safe? Is there a "maximal" γ-safe $\mathcal{R}$, in the sense that $\mathcal{R}$ permits any execution that is permitted by at least one γ-safe $\mathcal{R}'$?

It can be shown that $\mathcal{R}_1$ is maximal γ_1-safe for `Purchasing`.

EXAMPLE 2.4. Consider now the goal

$$\gamma_2 = \gamma_1 \\ \wedge \texttt{DEF(execApproved)} \rightarrow \texttt{bid} \geqslant 300 \qquad c3$$

Intuitively, clause $c3$ states that executives do not want to waste their time thinking about purchase requests where the `bid` is under 300. There are a variety of different ways that an execution can be γ_2-safe, but what happens if request has `prodType` = "hw" and has `bid` between 100 and 300? It may arise that `Estimate` will assign the `profitMargin` to be $\leqslant 10\%$, in which case `approved` will be assigned $false$. Because of clause $c3$ in γ_2, the execution does not satisfy γ_2, nor can it be extended to satisfy γ_2. We call such executions *dead-end*.

Consider now Figure 4, ignore the service labeled `EXC`, and focus on the ruleset $\mathcal{R}_{safe}$. Note that the rule permitting entry into `Estimate` (and effectively, into processing by the workflow) prevents input executions where the `prodType` = "hw" and the `bid` is strictly between 100 and 300. It can be shown that $\mathcal{R}_{safe}$ is a maximal γ_2-safe ruleset. ∎

Intuitively, the condition of the rule in $\mathcal{R}_{safe}$ for `Estimate` is "conservative", in the sense that if an input artifact o violates that condition, then o *may* lead to a dead-end execution. We now consider a more "lenient" form of maximality, in which executions are permitted to proceed through a workflow until the set of defined attribute values is sufficient to imply that the execution will become dead-end. To provide a formal setting for this, we introduce a new, "universal" *exception* service, denoted `EXC`. This has precondition $\neg\texttt{DEF(exc)}$ and has exactly one conditional effect, which is $true \rightarrow \texttt{exc} = true$. Intuitively, if a workflow execution includes the service `EXC`, then we shall view that execution as having moved into exceptional treatment. We do not consider recovery from exceptions in this paper.

Given a pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$, the *lenient* (also called *exception-permitting*) extension of $\mathcal{P}$, denoted $\mathcal{P}^{\texttt{EXC}}$, is the pair $(\mathcal{A}^{\texttt{exc}}, \mathcal{S}^{\texttt{exc}} \cup \{\texttt{EXC}\})$ where

(a) $\mathcal{A}^{\texttt{exc}}$ is obtained by adding, to each artifact class of $\mathcal{A}$, the new attribute `exc`, and

(b) $\mathcal{S}^{\texttt{exc}}$ is the result of modifying each service σ in $\mathcal{S}$ into the service $\sigma^{\texttt{exc}}$, which is obtained by replacing the pre-condition ρ of σ by $\rho \wedge \neg\texttt{DEF(exc)}$.

Intuitively, $\mathcal{P}^{\texttt{EXC}}$ extends (each artifact class of) $\mathcal{P}$ with the new exception service `EXC`, and modifies the pre-condition of each service σ of $\mathcal{P}$ so that it cannot be used if attribute `exc` has been defined. The `EXC` service acts as a sink – once the execution of an artifact o executes the `EXC` service, then attribute `exc` for o takes the value $true$, and no other service can be executed for o.

Given pre-workflow $(\mathcal{A}, \mathcal{S})$ and goal γ, a ruleset $\mathcal{R}$ is said to be γ-safe for $\mathcal{P}$ *with exceptions* if $\mathcal{R}$ is γ'-safe for $\mathcal{P}^{\texttt{EXC}}$ relative to $\mathcal{P}^{\texttt{EXC}}$, where γ' is the formula $\gamma \vee (\texttt{exc} = true)$.

It is trivial to construct rulesets that are γ-safe with exceptions; just route every execution to the `EXC` service. Informally, we say that ruleset $\mathcal{R}$ is maximal γ-safe with exceptions for $\mathcal{P}$ if (i) it is γ-safe for $\mathcal{P}^{\texttt{EXC}}$, (ii) each execution through $\mathcal{P}$ that satisfies γ is an execution through $\mathcal{R}$, and (iii) if in an execution $\vec{s}$ the defined

attributes of an artifact o imply that there is no extension of $\vec{s}$ that satisfies γ, then in $\mathcal{R}$ this execution is immediately moved to the EXC service. Intuitively, item (iii) states that in a γ-safe ruleset $\mathcal{R}$ maximal for exceptions, an execution that will fail is shifted to the EXC service at the earliest possible moment.

We can now phrase a third question.

Q3 (maximal ruleset γ-safe for $\mathcal{P}$ with exceptions:) Given pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ and goal γ, is there a ruleset $\mathcal{R}$ such that $(\mathcal{A}, \mathcal{S}, \mathcal{R})$ is maximal γ-safe?

EXAMPLE 2.5. Consider again Figure 4 but this time including the EXC service. Ruleset $\mathcal{R}_{safe\text{-}exc}$ shown there is maximal γ_2-safe for Purchasing with exceptions. To see this, note first that the Estimate service will be performed on all input artifacts, even the ones with prodType = "hw" and $100 <$ bid < 300. At the point of certainty that an execution will eventually become dead-end, however, that execution will be routed to the EXC service. In particular, consider a "hw" product with $100 <$ bid < 300, if the profitMargin computed by Estimate is $< 10\%$. It is easily inferred that such artifacts will eventually dead-end in Purchasing, and so ruleset $\mathcal{R}_{safe\text{-}exc}$ moves this artifact directly to EXC.

With $\mathcal{R}_{safe\text{-}exc}$, the routing to EXC is "eager", in the sense that a potential dead-end execution is diverted to the EXC service as soon as possible. An alternative would be "lazy" routing, i.e., to continue to perform services on the execution until a dead-end has actually been reached. In the example, this would happen if the entry condition for RoutineApproval in $\mathcal{R}_{safe\text{-}exc}$ were replaced by DEF(profitMargin), the edge from Estimate to EXC removed, the condition for entering ExecApproval were replaced by approved = $false \wedge$ bid $\geqslant 300$, and the condition for entering EXC were replaced by approved = $false \wedge$ bid < 300. ∎

We study here the case of building rulesets that enable "eager" routing to EXC, because these will have the effect of reducing the total cost of performing services before an execution ends up at EXC. Variations may also be relevant in practice.

We view questions **Q1**, **Q2**, and **Q3** as fundamental for goal-directed workflow construction and design-time analysis. Intuitively, **Q1** is asking whether it even makes sense to attempt to construct a workflow for goal γ. **Q2** focuses on construction of a workflow that permits all input artifacts which are guaranteed to support successful executions. If for a given context we cannot construct such a workflow, then probably we cannot use the goal-directed approach to workflow construction in that context. **Q3** provides a more robust approach to goal-directed workflow construction than **Q2**. By permitting exceptions, answers to **Q3** can guarantee that every execution through $\mathcal{P}$ that satisfies γ is supported. The workflows constructed for both **Q2** and **Q3** can give insights, at design time, about the properties of a goal γ, which can help the goal designer to understand whether the goal is capturing his/her intended semantics.

3. A MODEL FOR WORKFLOWS

In this section, we present several definitions that are needed to study the three workflow construction problems raised in the previous section.

The artifact workflow model follows the spirit of the model introduced in [6], however, it is tailored to the focus of the technical problems studied here. Notably, we focus here on workflows for a single artifact; we also leave out artifact states. On the other hand, we allow checking if an attribute has an assigned value and examining its value in service pre-/post-conditions and rule conditions. (Extensions for multiple artifacts will be discussed in Section 4.)

Let $\mathcal{L}$ be a first-order logic language and $\boldsymbol{S}$ be a first-order structure of $\mathcal{L}$ with a universe U. We assume some familiarity with standard logic notions (*formulas*, *sentences*, *quantifiers*, *satisfiability*, etc.) for $\mathcal{L}$ and $\boldsymbol{S}$. Intuitively, $\mathcal{L}$ and $\boldsymbol{S}$ are used to model individual data values in artifacts manipulated in workflows.

We assume the existence of the following two disjoint, countably infinite sets:

- **ATT** $= \{A, B, A_1, ...\}$ of *attributes*, and
- **SERV** $= \{\sigma, \sigma_1, ...\}$ of *service names*.

Without loss of generality, each attribute in **ATT** is also assumed to be a variable in $\mathcal{L}$, called an *attribute variable*. To deal with attributes with no assigned values, we extend the universe U with a special symbol "$\perp$".

DEFINITION. An *(artifact) schema* $\mathcal{A}$ is a finite set of attributes in **ATT**. An *artifact* of $\mathcal{A}$ is a mapping o from $\mathcal{A}$ to $U \cup \{\perp\}$.

Terms (over a schema $\mathcal{A}$) include variables in $\mathcal{L}$, constants in U, and for each attribute A in the schema $\mathcal{A}$, A and A' (the primed attribute A' is used to denote the new attribute value immediately after a service invocation). Similarly, if o is an artifact, we use o' to denote the artifact as the result of invoking a service on o.

Atomic formulas (over a schema $\mathcal{A}$) include $\text{DEF}(A)$, $\text{DEF}(A')$ where $A \in \mathcal{A}$ is an attribute variable, and atomic formulas built using predicates in $\mathcal{L}$ and terms in the standard manner. A formula is *attribute-only* if each variable occurring in it is an attribute variable or its primed version.

An *assignment* is a mapping from variables to U and (primed) attribute variables to $U \cup \{\perp\}$. Satisfaction of a formula (over $\boldsymbol{S}$) is defined in the standard manner, except that $\text{DEF}(A)$ ($\text{DEF}(A')$) is true if A (resp. A') has a value (i.e., $\neq \perp$). A pair of artifacts o, o' satisfies a formula φ under an assignment μ, denoted as $(o, o') \models \varphi[\mu]$ if $\boldsymbol{S} \models \varphi[\mu\langle o, o'\rangle]$ where $\mu\langle o, o'\rangle$ is an assignment modified from μ by mapping attribute variables to coincide with o and primed attribute variables to coincide with o'. When φ has no primed attribute variables, we also use "$o \models \varphi[\mu]$" to mean that o *satisfies* φ, or technically, $(o, o') \models \varphi[\mu]$ for every o'.

We next introduce the notion of a "service", which captures an available task that may be performed automatically or by humans. Services are the building blocks for assembling workflows. Since the focus of this paper is on workflow schemas that act on a single-artifact class, we also simplify the notion of a service; a more general notion can be found in [6].

DEFINITION. Let $\mathcal{A}$ be an artifact schema. A *service* over $\mathcal{A}$ is a tuple $(\sigma, R, W, \pi, \rho)$, where $\sigma \in$ **SERV** is a service name, R, W are finite sets of (respectively, read, write) attributes in $\mathcal{A}$, π and ρ are quantifier-free, attribute-only formulas representing the pre- and post-condition (resp.).

Given a service $(\sigma, R, W, \pi, \rho)$ and an artifact o, the *semantics* of

the service is intuitively defined as follows. If all attributes in R are defined in o and o satisfies π, the service σ may be invoked and if σ is invoked, its execution on o modifies o into o' such that (1) $A = A'$ for each attribute $A \in \mathcal{A} - W$, and (2) o' satisfies ρ (with a slight abuse of notation). In this case, we denote $o \vdash^{\sigma} o'$.

DEFINITION. A *pre-workflow (schema)* is a pair $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ where $\mathcal{A}$ is an artifact schema and $\mathcal{S}$ a finite set of services over $\mathcal{A}$. For a pair of artifacts o_1, o_2 of $\mathcal{A}$, o_1 *derives* o_2 *in* $\mathcal{P}$, denoted as $o_1 \vdash_{\mathcal{P}} o_2$, if $o_1 \vdash^{\sigma} o_2$ for some service $\sigma \in \mathcal{S}$.

From a pre-workflow schema, a workflow can be assembled by defining a set of "(business) rules". Roughly, a rule specifies which service is to be executed on which artifact and when.

We view the logic language $\mathcal{L}$ as a set of formulas. A *sub-language* of $\mathcal{L}$ is a subset of $\mathcal{L}$, e.g., the set of quantifier-free formulas in $\mathcal{L}$. In the following discussions, let $\mathcal{L}'$ be a sub-language of $\mathcal{L}$.

DEFINITION. An $\mathcal{L}'$*-condition* of an artifact schema $\mathcal{A}$ is a formula φ in $\mathcal{L}'$ with only attribute variables occurring free. An $\mathcal{L}'$-condition φ is *testable* if it can be determined whether o satisfies φ for each artifact o.

Consider a few candidates for the domain logic language $\mathcal{L}$, which were studied in constraint databases [30]: (1) dense or discrete total order (with $\leqslant$), (2) linear arithmetic, (3) real arithmetic (with $\leqslant$, $+$, $\times$), (4) integer arithmetic (with $\leqslant$, $+$, $\times$). Conditions for (1)-(3) are testable (with various complexity), while not testable for (4).

DEFINITION. Given a pre-workflow schema $\mathcal{P}$, an $\mathcal{L}'$*-rule* is an expression with the form "**if** φ **invoke** σ", where φ is a testable $\mathcal{L}'$-condition, and σ a service in $\mathcal{P}$. An artifact o_2 is *derived from* another artifact o_1 *using a rule* r, denoted as $o_1 \vdash^{r} o_2$, if $o_1 \models \varphi$ and $o_1 \vdash^{\sigma} o_2$. An $\mathcal{L}'$*-ruleset* is a finite set of $\mathcal{L}'$-rules.

As we noted in the previous section, when a rule is applicable in a situation, it does not have to be applied since only one of multiple applicable rules is applied. Such non-deterministic behaviors are often results of external decisions in business workflows.

DEFINITION. A *workflow (schema)* is a triple $\mathcal{W} = (\mathcal{A}, \mathcal{S}, \mathcal{R})$ where $\mathcal{P}{=}(\mathcal{A}, \mathcal{S})$ is a pre-workflow schema and $\mathcal{R}$ a finite set of rules over $\mathcal{P}$. $\mathcal{W}$ is said to *extend* $\mathcal{P}$. An artifact o_1 *derives* another artifact o_2 in $\mathcal{W}$, denoted as $o_1 \vdash_{\mathcal{W}} o_2$, if $o_1 \vdash^{r} o_2$ for some rule r in $\mathcal{R}$.

Let $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ be a pre-workflow schema. An *input (set)* for $\mathcal{P}$ is a subset $\boldsymbol{A}_{\text{init}} \subseteq \mathcal{A}$. In general, we shall consider a pre-workflow $\mathcal{P}$ and a given input $\boldsymbol{A}_{\text{init}}$. A *goal* for $\mathcal{P}$ is a satisfiable $\mathcal{L}'$-condition over $\mathcal{A}$.

A *path* for a pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ with input $\boldsymbol{A}_{\text{init}}$ is a sequence of artifacts $\vec{o} = o_1, ..., o_n$ where

1. $o_1 \models \text{DEF}(A)$ iff $A \in \boldsymbol{A}_{\text{init}}$ for each $A \in \mathcal{A}$,
2. For each $1 \leqslant i < n$, $o_i \vdash_{\mathcal{P}} o_{i+1}$.

Let γ be a goal. The path $\vec{o}$ is γ*-safe* if $o_n \models \gamma$.

Let $\mathcal{W} = (\mathcal{A}, \mathcal{S}, \mathcal{R})$ and $\boldsymbol{A}_{\text{init}}$ be an input parameter set for $(\mathcal{A}, \mathcal{S})$. A path $o_1, ..., o_n$ for $(\mathcal{A}, \mathcal{S})$ with input $\boldsymbol{A}_{\text{init}}$ is an *execution* of $\mathcal{R}$ if for each $i \in [1..(n-1)]$, $o_i \vdash^{r} o_{i+1}$ for some rule r in $\mathcal{R}$. We also refer to such as *executions* of $\mathcal{W}$.

4. WORKFLOW CONSTRUCTION

In this section, we formulate and study the workflow construction problems in a general context. We study Problems **Q2** and **Q3**; we give detailed definitions needed for Problem **Q2**, based on which it is relatively easy to formulate definitions for **Q3**. A key step of construction of a "maximally γ-safe ruleset" is the computation of "weakest pre-condition" of a service to guarantee that a given condition is satisfied after the service execution. We consider two kinds of weakest pre-conditions and formulate the computation process using quantifier elimination. In the following, we first present the weakest pre-condition and its computation, and then define and study the problems.

4.1 Weakest pre-conditions

First some technical considerations. We treat each (primed) attribute variable as a normal variable in the logic $\mathcal{L}$. However, since these variables may have undefined values, we use a distinct (new) variable for each of them and one fixed element in U which we name NULL to "encode" the undefinedness. For example, if A' is a primed attribute variable, $x_{A'} = \text{NULL}$ holds iff $\text{DEF}(A')$ is false.

For each formula φ in $\mathcal{L}$, we denote by φ^d the formula obtained from φ after the following substitutions: For each occurrence of an atomic formula θ involving an attribute variable A (or A'), (1) if θ is $\text{DEF}(A)$, it is replaced by $\neg(x_A = \text{NULL})$, (2) otherwise it is replaced by $\theta \wedge \neg(x_A = \text{NULL})$.

DEFINITION. The *core* of a service $(\sigma, R, W, \pi, \rho)$ is defined as $\mathit{core}(\sigma) = \langle R, W, \pi^d, \rho^d \rangle$.

For convenience, we denote $\mathit{core}(\sigma)$ as $\langle \pi(\bar{x}), \rho(\bar{x}\bar{y}) \rangle$ where $\bar{x}$ and $\bar{y}$ are enumerations of variables for attributes in R and primed attributes in W (resp.).

Similarly we replace a goal γ by γ^d but write it as $\gamma(\bar{x}\bar{y})$. We now define the notions of "weakest precondition" [15], which is a key to solve **Q2** and **Q3** for the service σ that will guarantee that the result of executing σ always satisfies the goal γ.

DEFINITION. Let $\langle \pi(\bar{x}), \rho(\bar{x}\bar{y}) \rangle$ be the core of a service σ and $\delta(\bar{x}\bar{y})$ a (sub-goal) condition. A $\forall$*-precondition of* σ, δ is a formula $\epsilon(\bar{x})$ such that ϵ logically implies π and $\boldsymbol{S} \models \forall\bar{x}(\epsilon(\bar{x}) \rightarrow \forall\bar{y}(\rho(\bar{x}\bar{y}) \rightarrow \delta(\bar{x}\bar{y})))$; an $\exists$*-precondition of* σ *and* δ is a formula $\xi(\bar{x})$ such that ξ logically implies π and $\boldsymbol{S} \models \forall\bar{x}(\xi(\bar{x}) \rightarrow \exists\bar{y}(\rho(\bar{x}\bar{y}) \wedge \delta(\bar{x}\bar{y})))$.

A $\forall$-(or $\exists$-)precondition is *weakest* if it is logically implied by every $\forall$-(or $\exists$-)precondition.

Let σ be a service and δ be a condition. We denote the weakest $\forall$-precondition (resp. weakest $\exists$-precondition) as $\text{WP}^{\forall}(\sigma, \delta)$ (resp. $\text{WP}^{\exists}(\sigma, \delta)$).

We note here that a weakest precondition in [15] corresponds to $\forall$-precondition in our setting. (The concept was formulated by Dijkstra in the programming language context, see details in Section 6. The notion of weakest precondition has also been used variously in AI and termed "regression" [42].)

LEMMA 4.1. For each service σ with core $\langle \pi(\bar{x}), \rho(\bar{x}\bar{y})\rangle$ and each condition $\delta(\bar{x}\bar{y})$,

$$\pi(\bar{x}) \wedge \forall\bar{y}(\rho(\bar{x}\bar{y}) \rightarrow \delta(\bar{x}\bar{y})), \quad \text{and}$$
$$\pi(\bar{x}) \wedge \exists\bar{y}(\rho(\bar{x}\bar{y}) \wedge \delta(\bar{x}\bar{y}))$$

are the weakest $\forall$-(resp. $\exists$-)precondition of σ and δ.

The lemma can be easily proved from the definitions.

4.2 Building maximal workflow schemas

As we shall see, $\forall$-preconditions can help to solve **Q2**, while $\exists$-preconditions can help to solve **Q3**. We now discuss Problem **Q2**.

DEFINITION. Let $\mathcal{P}$ be a pre-workflow, $\boldsymbol{A}_{\text{init}}$ an input, and $\mathcal{R}$ and $\mathcal{R}'$ sets of rules (in $\mathcal{L}'$) for $\mathcal{P}$. Then $\mathcal{R}$ *subsumes* $\mathcal{R}'$, denoted $\mathcal{R} \succeq \mathcal{R}'$, if each execution of $\mathcal{R}'$ over $\mathcal{P}$ and $\boldsymbol{A}_{\text{init}}$ is also an execution of $\mathcal{R}$.

Let γ be a goal. A path (execution) is γ-*dead-end* if it is not a prefix of any γ-safe path (resp. execution). When it is clear, we may use dead-end instead of γ-dead-end.

DEFINITION. Let $\mathcal{P}=(\mathcal{A}, \mathcal{S})$ be a pre-workflow, $\boldsymbol{A}_{\text{init}}$ an input set, and γ a goal for $\mathcal{P}$. An $\mathcal{L}'$-ruleset $\mathcal{R}$ for $\mathcal{P}$ is γ-*safe* if (1) each execution of $\mathcal{W}$ is γ-safe and (2) $\mathcal{W} = (\mathcal{A}, \mathcal{S}, \mathcal{R})$ has no dead-ends.

An $\mathcal{L}'$-ruleset $\mathcal{R}$ for $\mathcal{P}$ is *maximally γ-safe (in $\mathcal{L}'$)* if it is γ-safe, and for every $\mathcal{L}'$-ruleset $\mathcal{R}'$ for $\mathcal{P}$ that is γ-safe, $\mathcal{R} \succeq \mathcal{R}'$.

Q2: (Fix a language $\mathcal{L}' \subseteq \mathcal{L}$) Is there a maximally γ-safe ruleset $\mathcal{R}$ for $\mathcal{P}$ and $\boldsymbol{A}_{\text{init}}$? Is there an algorithm that constructs such a ruleset if it exists?

In the remainder of the section, we explore the problem **Q2** in the general setting in order to understand the theoretical underpinnings for the problem.

Consider a service $(\sigma, R, W, \pi, \rho)$ and a goal condition γ. In a naive approach, **Q2** could be solved with the following intuitive idea: search for a service σ_1 and compute the weakest $\forall$-precondition γ_1 of σ_1 for γ, and then search for another service σ_2 and compute the weakest $\forall$-precondition γ_2 of σ_2 for γ_1, and so on so forth until a precondition over the input set is obtained. At each step, each weakest $\forall$-precondition is used to create a rule.

In the following, we state that for the general case when $\mathcal{L}' = \mathcal{L}$, solving problem **Q2** is not easier than the complexity of the first order theory of the structure.

PROPOSITION 4.2. If Problem **Q2** is solvable for $\mathcal{L}$, then the first order theory of $\boldsymbol{S}$ is decidable.

Proof: (Sketch) Given a sentence φ in $\mathcal{L}$, we construct a service with no input/output and precondition and effects being true. Clearly the weakest $\forall$-precondition is the condition for the rule. The weakest $\forall$-precondition is true iff φ is true. ∎

We now consider $\mathcal{L}'$ to be the set of quantifier-free formulas of $\mathcal{L}$, denoted as $\mathcal{L}^{\text{QF}}$. In the following, we introduce an "invoke once" property of pre-workflows that express that the pre-workflow does not admit executions in which the same service is used more than once.

DEFINITION. A pre-workflow $\mathcal{P}$ is *invoke-once* if there is no sequence of services $\sigma_1, \ldots, \sigma_n$ for which there exists a path $o_1, \ldots, o_n$ of $\mathcal{P}$ such that $o_i \vdash^{\sigma_i} o_{i+1}$ for each $i \in [1..n-1]$ and for which there is some i, j, $i \neq j$, where $\sigma_i = \sigma_j$.

Speaking intuitively, there are many ways that a pre-workflow might "enforce" the invoke-once restriction. For example, in the restricted setting of Section 5, each service can write only one attribute, and can be invoked only if that attribute is currently undefined. As another example, the artifact schema might hold attributes which essentially perform book-keeping about which services have been invoked, and prevent multiple invocations.

THEOREM 4.3. For invoke-once pre-workflows, Problem **Q2** is solvable for $\mathcal{L}^{\text{QF}}$ if the first order theory of $\boldsymbol{S}$ is decidable and admits quantifier elimination.

Proof: (Sketch) Let $\mathcal{P}$ be an invoke-once pre-workflow. Since each service is executed at most once in executions of $\mathcal{P}$, consider all possible sequences of service invocations. For each such sequence, we compute weakest preconditions one by one starting with γ from the end of the sequence. At each step, a weakest $\forall$-precondition is used to form a rule in the ruleset, that allows to invoke the corresponding service. (See Algorithm 5.2 below for a more detailed description of a variant of this algorithm, that works in a specialized context.) Let $\mathcal{R}$ be the ruleset formed, and let $\mathcal{W}$ be the workflow created from $\mathcal{P}$ and $\mathcal{R}$.

To see that $\mathcal{R}$ is γ-safe and dead-end free, suppose that $o_1, \ldots, o_n$ is a path in $\mathcal{P}$ and $\mathcal{R}$ where o_1 is initial, $n > 1$, and o_n cannot be extended in $\mathcal{P}$ and $\mathcal{R}$. (We know that each path has bounded length because $\mathcal{P}$ is invoke-once.) We argue that o_n satisfies γ. Suppose otherwise. Consider the rule **if** α **invoke** σ that is used to move from o_{n-1} to o_n. Since this rule is in $\mathcal{R}$, there is some sequence $\sigma_1, \ldots, \sigma_m$ with weakest $\forall$-preconditions $\alpha_1, \ldots, \alpha_m$, and where $\alpha = \alpha_1$ and $\sigma = \sigma_1$. If $m = 1$, then $o_n = \sigma_1(o_{n-1})$ satisfies γ, a contradiction. So $m > 1$ and o_n satisfies α_2. Further, **if** α_2 **invoke** σ_2 is in $\mathcal{R}$. This means that σ_2 can be applied to o_n, that is, o_n can be extended after all, again a contradiction.

To see that $\mathcal{R}$ is maximal, suppose that there is some ruleset $\mathcal{R}'$ that is γ-safe and dead-end free. Suppose further that there is some initial artifact o for which there is an execution in $\mathcal{P}$ and $\mathcal{R}'$. where that execution uses the sequence $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$. Then o will satisfy the weakest $\forall$-precondition at the head of this sequence, and one can use the rules in $\mathcal{R}$ that were created for $\vec{\sigma}$ to move o to an artifact that satisfies γ. That is, the execution on o by $\mathcal{R}'$ is also an execution for $\mathcal{R}$. ∎

COROLLARY 4.4. *For invoke-once pre-workflows, Problem* **Q2** *is solvable for $\mathcal{L}^{\text{QF}}$ when $\mathcal{L}$ is (1) dense or discrete total order (with $\leqslant$), (2) linear arithmetic, (3) real arithmetic (with $\leqslant, +, \times$).*

PROPOSITION 4.5. If Problem **Q2** is solvable for $\mathcal{L}^{\text{QF}}$ then the first-order theory of $\boldsymbol{S}$ is decidable and admits quantifier elimination.

Proof: (Sketch) Consider a formula $\psi = \exists y \varphi(y, x_1, ..., x_k)$ where φ is quantifier free. We construct a service with core $\langle \textit{true}, \text{DEF}(y)\rangle$ and let $\varphi(y, x_1, ..., x_k)$ be the goal. Clearly, the weakest $\exists$-precondition is a quantifier-free formula equivalent to ψ. ∎

It is open if the converse of Proposition 4.5 is true in the general setting. The main difficulty is that the constructed workflow may have recursion, and thus the naive approach of performing an exhaustive search may not terminate.

4.3 Constructing workflows with exceptions

We now present a brief, informal discussion about an algorithm for resolving Problem **Q3** for the case of invoke-once pre-workflows. The key difference from **Q2** is that the use of exception services for bad cases means that it is not necessary to insist that *every* execution from a given input leads to a γ-safe endpoint. Thus, at the outer level **Q3** can be solved by using weakest $\exists$-preconditions. Importantly, a solution to **Q3** must also identify dead-end executions as quickly as possible, and direct them to EXC.

To provide a bit more detail, recall from Section 2 the extension of a pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ into a pre-workflow $\mathcal{P}^{\texttt{EXC}} = (\mathcal{A}^{\texttt{exc}}, \mathcal{S} \cup \{\texttt{EXC}\})$ with the EXC service. The algorithm used now is similar to the one described in the proof of Theorem 4.3 with one modification and an added step at the end. The modification, as noted above, is to use the weakest $\exists$-precondition rather than the $\forall$-precondition in the construction of the rules discussed in the proof. Let $\mathcal{R}_1$ denote the ruleset obtained from these. It can be shown that each γ-safe path of $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ is an execution of $\mathcal{R}_1$. However, there may also be dead-end executions permitted by $\mathcal{R}_1$. To satisfy the definition of maximal γ-safe with exceptions, we need to "shift" an execution to the EXC service as soon as it cannot be extended to be a γ-safe execution.

In general, an execution $o_1, \ldots, o_n$ is dead-end for a set of rules if no rule can be applied to o_n. By construction of $\mathcal{R}_1$, it is further true that for an execution $o_1, \ldots, o_n$, if there is not extension of this execution that reaches γ then the path is already dead-end. Thus, path $o_1, \ldots, o_n$ is dead-end for $\mathcal{R}_1$ iff o_n does not satisfy the condition of any rule in $\mathcal{R}_1$. Let Δ be the set of all conditions of rules in $\mathcal{R}_1$, and let $\varphi = \vee\{\delta \mid \delta \in \Delta\}$. Let α be the rule **if** $\neg\varphi$ **invoke** EXC, and let $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\alpha\}$. This rule has the desired effect of moving artifact instances to the EXC service once there is no possible way to continue execution to a point where γ is satisfied. Thus $\mathcal{R}_2$ provides a solution to **Q3** for $\mathcal{P}^{\texttt{EXC}}$ and γ.

As a final remark, we note that the technique described above can be applied to multiple artifacts with "fixed-link structures" [6]. Suppose further that the artifacts linked together have same (similar) lifespan. In this case, one can merge the artifacts into a single fat artifact and the algorithms for Problems **Q2**, **Q3** can be applied. It is interesting to explore situations when the linked artifacts have very different lifespans.

5. COMPLEXITY OF A SPECIAL CASE

We now consider the concepts presented above in a specialized context, which is both relatively simple and practically motivated. The context assumes that the underlying domain has just one sort, which is a dense linear order. It also imposes a number of restrictions on the services used, which in turn restricts how the services can be sequenced. The examples of Section 2 essentially lie within this context. In particular, the services of Section 2 do satisfy the restrictions. As for the underlying domain, the use of finite domains (e.g., Boolean) is easily simulated in the context of the dense linear order, and one can view the percentage and dollar amount domains as being dense.

We show here that even in the restricted context the problem of finding a maximal γ-safe ruleset is PSPACE-complete; develop a constructive algorithm for creating maximal γ-safe rulesets (both with and without exceptions) and consider its running time; and we describe some restrictions which can be used to reduce that running time. We discuss the collection of restrictions made in this previous section at length in Subsection 5.4.

5.1 A simple framework

This section focuses on a model called the *simple quantifier-free dense linear order artifact-based workflow framework*, and denoted $\mathbf{W}^{QF,<}$. This was illustrated in Section 2. As underlying first-order logic structure we assume a single sort (domain) which has a dense linear order, which we denote as $\mathcal{L}^{\leqslant}$. In $\mathbf{W}^{QF,<}$ it is assumed that each service writes exactly one attribute.

In $\mathbf{W}^{QF,<}$ the service pre-conditions and the conditions for rules are quantifier-free formulas in $\mathcal{L}^{\leqslant}$. The service post-conditions have the form of *conditional effects*, that is, each post-condition is a conjunction of formulas of form $\alpha \rightarrow \beta$ (as in the examples of Section 2). Further, each antecedent α is a quantifier-free formula in $\mathcal{L}^{\leqslant}$, and each β is an atomic expression having one of the following forms: $\texttt{DEF}(A)$ for attribute A; or $t_1 \circ t_2$ where exactly one of t_1, t_2 is the attribute A and the other is a constant, and where $\circ$ is one of $\{=, \neq, >, \geqslant, <, \leqslant\}$.

Our final restrictions focus on preventing arbitrary permutations of the service invocations, and thereby reducing somewhat the number of weakest preconditions that need to be computed. To begin, a service σ is *well-formed* if the pre-condition has form $\pi = \varphi \wedge \bigwedge\{\texttt{DEF}(B) \mid B$ occurs in the pre-condition or the antecedent of some conditional effect$\} \wedge \neg\texttt{DEF}(A)$, where A is the attribute defined by σ and φ is a quantifier-free formula over the read set of σ. In this case, we say that σ *requires* $\texttt{DEF}(B)$ for each of the B's mentioned. Further, if A is the attribute defined by σ, it is assumed for each artifact o_1 that satisfies π that (a) there is at least one artifact o_2 such that $o_1 \models^{\sigma} o_2$, and (b) all artifacts o_2 satisfying $o_1 \models^{\sigma} o_2$ have $o_2.A$ defined. (Intuitively, this says that if you can enter service σ then you will exit from it with a defined value for A.) Note that under these assumptions, at most one service that defines attribute A can be invoked in an execution. A pre-schema $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ is *well-formed* if each service in $\mathcal{S}$ is well-formed.

We say that a well-formed pre-schema $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ *admits an attribute ordering* if there is some ordering $A_1, ..., A_n$ of the attributes such that for each $i \in [1..n]$ and each service σ that defines attribute A_i, σ requires only attributes in $\{A_1, ..., A_{i-1}\}$. Such an ordering is called *admissible*. Each workflow in $\mathbf{W}^{QF,<}$ is assumed to admit ordering of attributes.

5.2 Complexity

We consider first the complexity of finding maximal rulesets in $\mathbf{W}^{QF,<}$.

Problem: Coverage by γ-safe Ruleset
Instance: A pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ in $\mathbf{W}^{QF,<}$, a fixed set of attributes to be initialized, and a goal γ. (The input to this problem is considered to have size equal to the length of the string of symbols used to specify all of $\mathcal{A}$, $\mathcal{P}$, and γ.)
Question: Is there a γ-safe ruleset $\mathcal{R}$ for $\mathcal{P}$ that permits all input artifacts to enter $(\mathcal{A}, \mathcal{S}, \mathcal{R})$.

THEOREM 5.1. The problem of Coverage by γ-safe Ruleset is PSPACE-complete.

Proof: Suppose that artifact schema $\mathcal{A}$, pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$. and goal γ are given. Let L denote the size of the input.

We show first that the problem is within PSPACE. We start by creating the algebraic cell decomposition [29] for $\mathcal{P}$ and γ. To this end, let $\mathcal{A} = \{A_1, \ldots, A_n\}$, let $\mathcal{C}$ be the set of constants appearing in $\mathcal{P}$ or γ, and let $c = |\mathcal{C}|$. Assignments[1] ν and μ of the attributes in $\mathcal{A}$ to the underlying domain $U \cup \{\perp\}$ are said to be *equivalent* if for each $i, j \in [1..n]$ and constant $d \in \mathcal{C}$, we have (a) $\nu(A_i)$ is undefined iff $\mu(A_i)$ is undefined, (b) $\nu(A_i) \leqslant d$ iff $\mu(A_i) \leqslant d$, (c) $\nu(A_i) \geqslant d$ iff $\mu(A_i) \geqslant d$, and (d) $\nu(A_i) \leqslant \nu(A_j)$ iff $\mu(A_i) \leqslant \mu(A_j)$. A *cell* is an equivalence class of assignments. Each cell can be specified by writing a sequence of all attributes and constants, separated by either $<$ or $=$, where they appear in the sequence according to their ordering as in some (any) assignment in the equivalence class identified by the cell. That is, each cell can be specified in length which is in $O(c + n)$ and thus in $O(L)$.

As an aside, we note that the total number of cells is in $O((c+n)^n)$. To see this, imagine assigning each attribute A_i to a location in the sequence that represents a cell. For the first attribute there are $2c + 1$ choices, and for the k^{th} attribute there are no more than $2(c + (k - 1)) + 1$ choices.

A *symbolic execution* through $\mathcal{P}$ is an execution where cells rather than artifacts are used. It is easily verified that there is a natural homomorphism (bisimulation) from true executions to symbolic executions, and that to check for the existence of the γ-safe ruleset we need consider only rulesets using constants occurring in $\mathcal{S}$ and γ, and consider only symbolic executions.

We shall describe a non-deterministic PSPACE computation that determines whether there is a γ-safe ruleset $\mathcal{R}$ for $(\mathcal{P}, \mathcal{S})$ that permits all input artifacts to enter $(\mathcal{A}, \mathcal{S}, \mathcal{R})$. From this, we can apply Savitch's theorem to conclude that this test can be performed in deterministic PSPACE. While the test for existence of a γ-safe ruleset is within polynomial space of L, the ruleset itself might have size on the order of the number of cells in the algebraic cell decomposition.

The computation shall be guided by a tree T that is described now. Intuitively, this is a tree with alternating levels of 'and' and 'or' nodes, where the children of 'and' nodes are labeled with selected cells and the children of 'or' nodes are labeled with selected services from $\mathcal{S}$. Let $B_1, \ldots, B_k, C_1, \ldots, C_m$ be an admissible ordering of the attributes in $\mathcal{A}$, where the attributes $B_1, \ldots, B_k$ are the initialization attributes for $\mathcal{P}$.

The tree T has height $2m + 1$. For a node v of T, we use $v.cell$ to denote the cell labeling v (if there is one), and $v.service$ to denote the service labeling v (if there is one).

The root of T (level 0) is an 'and' node, and has a child for each cell α which has values for $B_1, \ldots, B_k$ and has null for each of $C_1, \ldots, C_m$. Given a node v at an odd level $i = 2j + 1$, $j \in [0..m - 1]$, v is an 'or' node. Further, there is a child of v in T for each service $\sigma \in \mathcal{S}$ such that (a) σ defines attribute C_{i+1}, and (b) $v.cell$ satisfies the pre-condition of σ. If v is at an even level $i = 2j$, $j \in [1..m]$, then v is an 'and' node. Further, there is a child of v for each cell α that (a) has values for $\{B_1, \ldots, B_k, C_1, \ldots, C_j\}$ and null for the remaining attributes, and (b) that could result from the application of $v.service$ on $v'.cell$, where v' is the parent of v. It can be verified that there is a γ-safe ruleset $\mathcal{R}$ with the desired properties iff the following holds, where N_i denotes the set of nodes of T at level i, $i \in [1..2m + 1]$:

$$
\begin{array}{ll}
(\dagger) & \forall v_1 \in N_1 \ \exists v_2 \in N_2 \ \forall v_3 \in N_3 \ldots \\
& \quad \exists v_{2m} \in N_{2m} \ \forall v_{2m+1} \in N_{2m+1} \\
& \quad [v_{i+1} \text{ is a child of } v_i \text{ for each } i \in [1..m-1], \text{ and} \\
& \quad v_{2m+1}.cell \text{ satisfies } \gamma]
\end{array}
$$

The tree T can now be used to guide a non-deterministic computation, in space polynomial of L, to check property (†). At each step, the computation will hold one partial path from the root, including the services and cells associated with the nodes along that path. The computation performs a depth-first search across all relevant paths. For a path p of length $i = 2j + 1$, $j \in [0..m - 1]$, for the first entry into node p_i one child of p_i (i.e., one service defining C_{j+1} that has true pre-condition for $p_i.cell$) is non-deterministically chosen and recorded. This is because p_i is an 'or' node. Given a path p' of length $i = 2j$, $j \in [0..m]$, each child of p'_i must be processed (i.e., the sub-trees below have to be processed), because p'_i is an 'and' node. If $j = m$, then each child of $p'_i.cell$ must satisfy γ. It can be verified that there is a successful non-deterministic computation iff property (†) holds. Note that the amount of storage required is on the order of the space needed to store m assignments, which is at most a quadratic of L. This completes the argument that the Coverage problem is within PSPACE.

We now show that the problem is PSPACE-hard. We perform a reduction from Quantified Boolean Formulas (QBF). Let $\alpha = Q^1 x_1 Q^2 x_2 \ldots Q^n x_n \beta(x_1, \ldots, x_n)$, where each Q^i is either $\forall$ or $\exists$, and $\beta(x_1, \ldots, x_n)$ is a Boolean formula over $x_1, \ldots, x_n$.

We shall build a pre-workflow $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ and goal γ that is related to α. Interestingly, we shall need only one constant in this pre-workflow. It will be shown that α is valid iff there is a γ-safe ruleset that permits all inputs to enter into an execution. Thus, α is valid iff each maximal γ-safe ruleset permits all input artifacts to enter into an execution.

For the artifact $\mathcal{A}$, we have attributes

$$
\begin{array}{ll}
a_i & i \in [0..n] \\
b_i, c_i & i \in [1..n] \text{ and } Q^i = \exists
\end{array}
$$

Attribute a_0 will act as the input parameter, and is ignored during the execution of all services of the workflow.

For $i \in [1..n]$, attribute a_i will correspond to variable x_i. The b_i's and c_i's will be used for bookkeeping. We allow 0 (zero) to be a constant in the domain of these attributes. Intuitively, if during execution we have $a_i > 0$, this will correspond to x_i being assigned the value $true$, and if $a_1 \leqslant 0$ this will correspond to x_i being assigned the value $false$.

We use the following family of services

$\sigma_i : a_i$ for $i \in [1..n]$ where $Q^i = \forall$
pre-condition: $\mathsf{DEF}(a_{i-1})$
post-condition: $true \rightarrow \mathsf{DEF}(a_i)$

[1]The reader will note that term 'assignment' used here is actually co-extensive with the term 'artifact'. We use 'assignment' to avoid confusion, since the way assignments are used in this proof is quite different than how artifacts are used in workflow executions.

$\tau_i^{\mathrm{T}} : b_i$ for $i \in [1..n]$ where $Q^i = \exists$
  pre-condition: $\mathtt{DEF}(a_{i-1}) \wedge \neg\mathtt{DEF}(c_i)$
  post-condition: $true \rightarrow \mathtt{DEF}(b_i)$

$\tau_i^{\mathrm{F}} : c_i$ for $i \in [1..n]$ where $Q^i = \exists$
  pre-condition: $\mathtt{DEF}(a_{i-1}) \wedge \neg\mathtt{DEF}(b_i)$
  post-condition: $true \rightarrow \mathtt{DEF}(c_i)$

$\sigma_i : a_i$ for $i \in [1..n]$ where $Q^i = \exists$
  pre-condition: $\mathtt{DEF}(b_i) \vee \mathtt{DEF}(c_i)$
  post-condition: $\mathtt{DEF}(b_i) \rightarrow a_i > 0$
  $\mathtt{DEF}(c_i) \rightarrow a_i \leqslant 0$

It is easily verified that the pre-workflow just described is in $\mathbf{W}^{QF,<}$.

Intuitively, if $Q^i = \forall$, then σ_i can yield a value bigger or less than 0 for a_i, i.e., it can "produce" the value true or false for x_i. If $Q^i = \exists$, then the rules can be used to determine which of τ_i^{T} or τ_i^{F} can be invoked (and only one of them will succeed in any case). This in turn permits the rules to "control" whether x_i becomes true or false.

Assume that in β there is no negation except on variables. We construct γ from β by replacing each non-negative term x_j in β by $a_j > 0$, and each negative term $\neg x_j$ in β by $a_j \leqslant 0$.

Suppose first that α is not valid. This means that $\neg\alpha$ is satisfiable. Let $\mathcal{R}$ be a maximal ruleset for $\mathcal{P}$ and γ. Suppose that $\mathcal{R}$ permits some execution. This means that for at least one value of a_0, an execution can be started for $\mathcal{R}$. Using the facts that $\neg\alpha$ is satisfiable, that $\mathcal{R}$ does not permit dead-end executions, and that the services σ_i can yield any output, it is now possible to construct an execution valid for $\mathcal{R}$ which ends with all of the a_i's defined, but with γ not satisfied. This is a dead-end execution after all, a contradiction.

Suppose now that α is valid. We shall construct a ruleset $\mathcal{R}$ for $\mathcal{P}$ that is γ-safe and that permits all input artifacts to enter the system. Let $i \in [1..n]$ be fixed, where $Q^i = \exists$. If ν is a truth assignment for $\{x_1, \ldots, x_{i-1}\}$, let $\nu + \{x_i/true\}$ denote the extension of ν where x_i is assigned to $true$, and define $\nu + \{x_i/false\}$ analogously. Create a Boolean formula $\varphi_i^T(x_1, \ldots, x_{i-1})$ such that

$$\varphi_i^T[\nu] = true \text{ iff}$$
$$Q^{i+1}x_{i+1} \ldots Q^n x_n \beta(x_1, \ldots, x_n)[\nu + \{x_i/true\}] = true$$

Analogously, create $\varphi_i^F(x_1, \ldots, x_{i-1})$ such that

$$\varphi_i^F[\nu] = true \text{ iff}$$
$$Q^{i+1}x_{i+1} \ldots Q^n x_n \beta(x_1, \ldots, x_n)[\nu + \{x_i/false\}] = true$$

In general, we will not have $\varphi_i^F \equiv \neg\varphi_i^T$. However, suppose that ν is chosen so that $Q^i x_i \ldots Q^n x_n \beta(x_1, \ldots, x_n)[\nu]$ is valid. (This is trivial for the case for $i = 1$, since α is assumed valid.) Since $Q^i = \exists$, at least one of $\varphi_i^T[\nu]$ or $\varphi_i^F[\nu]$ is true.

Assume that φ_i^T and φ_i^F have no negations except on variables. Let $\hat{\varphi}_i^T(a_1, ..., a_i)$ be constructed from φ_i^T by replacing each non-negative term x_j by $a_j > 0$ and each negative term $\neg x_j$ by $a_j \leqslant 0$. Construct $\hat{\varphi}_i^F(a_1, ..., a_i)$ analogously.

Build the ruleset $\mathcal{R}$ as follows.

for $i \in [1..n]$ with $Q^i = \forall$, include
  r_i = *if* $\mathtt{DEF}(a_{i-1})$ *then* σ_i
for $i \in [1..n]$ with $Q^i = \exists$, include
  r_i^T = *if* $\mathtt{DEF}(a_{i-1}) \wedge \hat{\varphi}_i^T(a_1, \ldots, a_{i-1})$ *then* τ_i^T
  r_i^F = *if* $\mathtt{DEF}(a_{i-1}) \wedge \neg\hat{\varphi}_i^T(a_1, \ldots, a_{i-1})$
  $\wedge\, \hat{\varphi}_i^F(a_1, \ldots, a_{i-1})$ *then* τ_i^F

It is straightforward to show that if α is valid, then on any input there are successful executions that satisfy γ, and no dead-end executions. ∎

5.3 A constructive algorithm

This subsection describes a constructive algorithm for building maximal γ-safe rulesets for $\mathbf{W}^{QF,<}$ pre-workflows. This is described so that we can discuss a contributor to the high running time of the ruleset construction, and identify restrictions that reduce that running time.

Recall that pre-workflows in $\mathbf{W}^{QF,<}$ admit ordering of the attributes. Let $A_1, \ldots, A_n$ be one such ordering. In this case, to perform the construction of rules as in the proof of Theorem 4.3, we can focus on a directed graph which has n layers (where n is the number of attributes). The i^{th} layer will consist of all services that define attribute A_i, and for each $i \in [1..n]$ edges are included from each service at layer $i-1$ to each service at layer i. As will be seen, although the full number of paths in such a graph is exponential in n, the number of rules needed to solve **Q2** (or **Q3**) is equal to the number of services.

The main algorithm is now given. In the algorithm it is assumed that the weakest pre-condition formulas that are computed are transformed to be quantifier-free.

ALGORITHM 5.2. Maximal γ-safe ruleset for $\mathbf{W}^{QF,<}$
Input: $\mathcal{P} = (\mathcal{A}, \mathcal{S})$ and γ in $\mathbf{W}^{QF,<}$,
and initialization attributes $\mathcal{I}$
Output: maximal γ-safe ruleset $\mathcal{R}$ for the above
Procedure:
1. Assume that $A_1, \ldots, A_n$ is an ordering of the
2. attributes so that each service defining A_i requires
3. only attributes from $A_1, \ldots, A_{i-1}$.
4. For each service σ that defines A_n, create the
5. rule R_S = **if** $wp^{\forall}(\sigma, \gamma)$ **then invoke** σ.
6. Do for each $i \in [1..n-1]$ in reverse order:
7. Let $\sigma_1, \ldots, \sigma_k$ be the services that define A_{i+1},
8. and let R_j = **if** α_j **then invoke** σ_j be the
9. rule already constructed by the
10. algorithm for σ_j
11. For each service σ that defines A_i, create the
12. rule R_σ = **if** α **then invoke** σ where
13. α = the $wp^{\forall}$ of σ and the disjunction of
14. the formulas α_j, $j \in [1..k]$.

It is straightforward to verify that this algorithm is correct.

We have developed constructive algorithms for computing quantifier-free formulas equivalent to $wp^{\forall}(\sigma, \varphi)$ and $wp^{\exists}(\sigma, \varphi)$; these involve a detailed analysis of the formula φ and the pre- and post-conditions of σ, which essentially comes down to removal of existential quantifiers.

As noted previously, Algorithm 5.2 constructs one rule for each service in $\mathcal{P}$. The astute reader will have noticed, however, that there is the potential for the size of the conditions in the rules to grow to be size exponential in the number of attributes. This is because the condition for the rule of a service defining A_i is based on a conjunction of formulas, one for each service that defines A_{i+1}.

We now explore the question of whether further restrictions on the pre-workflows in $\mathbf{W}^{QF,<}$ can help to limit the size of the conditions in the rules, thereby reducing the overall running time and space of the algorithm. A key factor is the number of attributes that are involved in the construction of $wp^{\forall}$ formula α in line 14 of the algorithm.

Suppose that the σ in the loop of line 11 is at level i, let $p > i$, and suppose that σ' is a service at level p. If A is an input attribute for σ', and $A = A_q$ for some $q < i$, then A can occur in at least one of the formulas α_j mentioned in line 14. Attributes A from γ that are defined above level i can also occur in one of the α_j's.

More generally, define *free*(i) to be $\{A_q \mid q < i$ and A_q appears in γ or is an input attribute for some σ_p with $p > i\}$, and define *width*$(i) = |$*free*$(i)|$. Let w = *max*$\{$*width*$(i) \mid 1 \in [1..n]\}$. It can be shown that the time required to compute $wp^{\forall}[\sigma, \alpha]$ in line 13 is bounded by some constant $c = f(b)$, where f is no more than doubly exponential in b times (b plus the number of constants appearing in $\mathcal{S}$ and γ). (The latter multiplicand is included because it helps to control how many atoms can be constructed using the variables in α.)

Intuitively, the value of *width*(i) corresponds rather coarsely to the amount of "scratch paper" that an execution needs to retain just after computing the attribute A_i. A pre-workflow with small maximum width is one that produces attribute values and then consumes them (for the last time) fairly quickly, and a pre-workflow with large maximum width is one that will produce many attribute values before it uses them for the final time. The discussion above indicates that for pre-workflows that have a very small maximum width the computation of the maximal γ-safe ruleset can be performed with runtime at a smaller exponential than for the pre-workflows which have high maximum width.

5.4 Discussion

This subsection briefly reviews the restrictions made for the results obtained in this section, considers how realistic they are, and what might be involved in relaxing them. The key assumptions for the technical results are (a) focus on a single artifact type; (b) each service writes exactly one attribute; (c) services are well-formed; and (d) the pre-workflow has an admissible attribute ordering. The combination of (b) and (c) imply an "invoke-once" semantics for services, that is, each service can be invoked at most once in an execution. Restriction (c) also implies a "write-once" or "monotonicity" property, i.e., that each attribute can be written at most once.

Consider first the focus on single artifacts. A key consideration here is the "life expectancy" of an artifact [8], e.g., some artifacts, such as *order* might have a life expectancy of days or weeks, where as other artifacts, such as *customer*, which would focus on the overall experience of customers over time, might have a life expectancy of years. The results of this section can be extended to work with a bounded set of artifact types that have a similar life expectancy, by simulating them with a single "super"-artifact, containing all attributes contained by the original artifacts (some renaming may be necessary). For example, this approach is discussed using the notion of "fixed-link structures" in [6], and also used in [14]. However, if two artifact types have substantially different life expectancy, then there will typically be many artifacts of one type for just one artifact of the other type. In this case, the "super"-artifact would essentially involve a set-valued attribute, and as seen in [14] this can significantly complicate the formal properties of the model. It remains open what restrictions would need to be made on set-valued attributes and/or artifacts with differing life expectancies in order to achieve results analogous to the ones obtained in this section.

Restriction (b) alone, that services write just one attribute, is included mainly for technical convenience. For example, if a service may write more than one attribute, then complex tautological reasoning is sometimes required when performing an optimized computation of the weakest precondition. Intuitively, this is because certain combinations of effects may lead to tautologies (which would lead to satisfiable weakest preconditions), even when the weakest precondition for each affected sub-formulae in isolation would be equivalent to $false$. In general, restriction (b) can be dropped without impacting the PSPACE result.

The well-formedness restriction (c) is quite strong; together with (b) it implies that each service can be executed at most once. The restriction enables Algorithm 5.2 to be relatively straightforward. If multiple invocations of a service were permitted, then executions could potentially have arbitrary length. Although the overall memory of the system at each point in time is bounded (by the space available in the artifact for storing values), it is unclear how the algorithm could be generalized to this case. Further, it remains open whether the PSPACE upper bound of Theorem 5.1 will hold in the context of multiple service invocations. (In principle, one might have to keep track of computations whose length is on the order of the number of cells in the algebraic cell decomposition.)

Restriction (d) concerning admissible ordering of the attributes is also quite strong. It implies that the attributes can be organized into a DAG, and that service invocations must be sequenced according to some topological sort of the DAG. This assumption significantly simplifies both the proof of the PSPACE upper bound and the construction of Algorithm 5.2. However, the DAG assumption does not appear to be critical to either of these, as long as each service is known to be invoke-once.

We finally consider whether the results obtained in this section can be applied in practice. On the one hand, experience in the field [5, 7] has shown that there are typically artifact types of different life expectancies, and that there are often cycles in the lifecycle of an artifact. Assuming that we focus on a single artifact, then it is often the case that the cycles are "local", and can be encapsulated to be viewed as essentially a single service. Many services are invoke-once, and in many situations the high-level flow of services in a lifecycle is DAG-based. Further, at IBM Research there is active work on developing a hierarchical approach to specifying lifecycles, where the work units in a given level of the hierarchy can be organized in either a declarative or more procedural manner. This suggests that the results developed here might be applied directly in selected contexts within a practical system. It is clear, however, that much research remains to be done.

6. RELATED WORK

We describe here related work in the following areas: the use of weakest pre-conditions in formal verification and synthesis, verification of artifact-based workflow, constraint-based systems, semantic web services, AI planners for workflow synthesis, and decision theory as found in AI and operations research.

The notion of weakest condition, which originated in [23] and subsequently studied in [15], has been used widely in formal verification and the theory of program derivation (synthesis). In verification [12], algorithms have been developed to deal with programming constructs as arise in general programming languages. The problems studied are thus quite general, and typically have high complexity. Techniques such as BDDs help with practical situations. With regards to program derivation, work such as [34, 36] build upon and extend the initial ideas in [15] to develop a general theory for derivation of programs involving the assignment statement and control structures. In contrast, the services in our work are more general than assignment statements (which can be viewed as services of specific types) and we distinguish $\forall$-preconditions and $\exists$-preconditions. Also, we focus on the complexity aspect which the earlier work did not address. On the other hand, the techniques for reasoning through control flow constructs in their work may have a potential use in workflow construction, but this is not exactly clear due to the different style of "programming" in the declarative workflow setting. One interesting note is that the avoidance of "dead-end" seems to resemble Dijkstra's concept of "free of miracles" [34, 36].

Work on formal analysis of artifact-centric business processes in restricted contexts has been reported in [6, 18, 19]. Properties investigated in these studies include reachability [18, 19], general temporal constraints [19], and the existence of complete execution or dead end [6]. Citations [18, 19] are focused on an essentially procedural version of artifact-centric workflow, and [6] is the first to study a declarative version, albeit at a very abstract level in which attributes are either defined or undefined (i.e., the values associated with attributes are not considered in the formalism). Both [14] and the current paper extend that model by letting attribute values range over a dense linear order, and incorporate non-deterministic services with pre-conditions and conditional effects specified in terms of the underlying domain. Unlike the current paper, [14] permits set-valued attributes, interaction of multiple artifacts (in limited settings), and a static external database that services can refer to. That paper studies static analysis of properties specified in a first-order extension of linear temporal logic, and maps the boundaries of decidability.

A key technical ingredient in this paper is the use of constraints, e.g., in modeling and reasoning about services. Constraints have been used in logic programming [27], databases [28], and verification (e.g., of hybrid systems [16]). In particular, our model of services is reminiscent of constraint relations [28, 22, 30]. However, one key difference is that the semantics are very different: a constraint relation represents a set of points, while the pre- and post conditions specify the set of possible input-output pairs. Also, it is typical to manipulate constraint relations using first-order queries, whereas the pre- and post-conditions of services are used to ensure proper chaining of the services.

The declarative artifact-centric workflow model of the current paper can be viewed as a specialized form of semantic web services as in OWL-S [13] and subsequent efforts [2, 44]. In particular, the use of non-determinism and pre- and post-conditions to characterize services is closely related to the use of Input, Output, Precondition, conditional Effect (IOPE) in OWL-S. Similar to the current paper, [35] develops an approach to create a composition of semantic web services that satisfies a static goal; that work uses a construction based on Petri nets. Another approach to automated composition of web services is provided by the Roman model and follow-up work [4, 17, 3, 39]. The Roman model focuses purely on data-less activities, and the complexity of the composition problem is in EXPTIME [4, 17]. The Colombo model can be viewed as a substantial extension of the Roman model, which incorporates data into a formal semantic web services framework in a rich way [3]. However, the complexity of the composition problem in Colombo is EXPSPACE. Thus our PSPACE results for workflow construction in the presence of data bring us one step closer to being practical.

Turning to the use of AI techniques in automated workflow design, [33] describes a system that uses a partial-order planner based on UCPOP [38] to construct business processes. In that work, a plan, which is essentially a use-once workflow schema, is built on-demand for each individual set of inputs. While there are some implicit conditionals in the plans, they have little flexibility once an execution has started. Our approach, on the other hand, generates a general-purpose workflow schema that can be used for all possible inputs, and is least restrictive in the sense that it leaves maximal flexibility to its user. For example, the precise scheduling of services is not fixed in advance, and could be varied to achieve certain optimizations.

The planning approach to workflow design described in [40] uses weakest pre-conditions (called "regression" there) in a manner similar to the current paper. This work starts with a desired goal and a family of "actions" (which are similar to our services but deterministic), and uses weakest pre-conditions to obtain a tree of all possible ways the goal can be achieved. This tree is used to generate a policy, that is, a mapping from conditions to actions, where each condition corresponds to a different possible state of the world. It is shown in [20] that the worst case complexity is linear in the size of the state space, which is exponential in the number of attributes used in the environment. Our approach differs by permitting a dense linearly ordered domain and non-deterministic services, and by building a general-purpose workflow schema. Further, we have identified some cases where our approach appears to be pragmatically feasible.

Finally, we note that decision theoretic approaches to similar problems can be found in AI and operations research, e.g. [9, 10]. These differ from our perspective in that they consider stochastic uncertainty, and also explicit costs and rewards of actions. In that framework, the objective is to maximize the expected accumulated reward.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we formulate the technical problem of goal-directed workflow construction in the context of declarative artifact-centric workflow, and develop results concerning the general setting, design-time analysis, and the synthesis of workflow schemas from goal specifications. The work is among the important initial steps along the path towards eventual support for tools that enable substantial automation for workflow design, analysis, and modification.

In terms of future work, one area is to consider the same problems but with different settings. For example, how do we handle

recursive workflows, i.e., not satisfying "invoke once". How do we generalize to cases with multiple artifacts which are linked together but have very different life expectancies? To what extent can we permit set-valued attributes? What if different underlying domains are used (e.g., the integers)? What if limited quantification is permitted in the specification of services or rules? As noted in Section 5, there is currently active work at IBM Research on incorporating hierarchy into lifecycle specifications in the artifact-centric approach; perhaps this can be leveraged to reduce the complexity of automated workflow synthesis.

A rich area is to extend the goals used here to include temporal operators. We expect that traditional approaches (e.g., [1, 11]) could be used to transform temporal goals into end-state goals such as used here, perhaps with the overhead of adding new attributes to the artifact type. It is unclear how many attributes would be needed, and how temporal goals would impact results such as the PSPACE completeness.

Another direction concerns whether incremental modifications to goals could be translated in an incremental fashion into new workflow schemas. This would be especially useful in the context of workflow evolution, in cases where relatively modest modifications are being made. Separate techniques might be useful for the somewhat distinct cases of adding/removing attributes from the artifacts and modifying the business goals or constraints about how or when certain attributes are to be computed.

In the model used in this paper, the services read only from artifacts and modify only artifacts. In the general case, of course, workflow services may use or impact the "outside world". It will be interesting to explore how the results and techniques obtained here can be generalized to the case with external databases that can be read and/or updated, and other side-effecting operations.

A final promising area is to see whether techniques developed in the current paper have relevance in the area of execution optimization of declarative artifact-centric workflow schemas. As discussed in [8], it appears useful to structure the implementation and optimization problem in three layers: conceptual layer (the rules-based workflow schemas generated in the current paper), logical implementation layer, and physical implementation layer. (These layers are analogous to relational database query implementation, with SQL at the conceptual layer, the relational algebra at the logical implementation layer, and the details of indexing and relational operator implementations at the physical implementation layer.) Indeed, it appears that the conceptual model described in [37] might provide a starting point for the logical implementation layer. It would be an interesting exercise to "compile" the rules of a declarative artifact-centric workflow schema into artifact workflows in the model of [37], which can then be implemented in a fairly direct manner. Optimization criteria, e.g., around resource limitations or the prioritization of particular artifact instances, could be incorporated into this compilation process. Techniques from [25] and from AI planning may also be relevant here.

Acknowledgments

The authors are grateful to K. Bhattacharya for fruitful discussions, and benefited from conversations with A. Nigam, N. Caswell, and F. Wu. The authors also thank several anonymous referees for suggesting a number of improvements in the exposition. The research reported on here supported in part by NSF grants IIS-0415195, CNS-0613998 and IIS-0812578.

8. REFERENCES

[1] J. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI06)*, pages 788–795, Boston, MA, July 2006.

[2] S. Battle et al. Semantic Web Services Ontology (SWSO) Version 1.0. `http://www.daml.org/services/swsf/1.0/swso/`, May 2005.

[3] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proc. 31st Int. Conf. on Very Large Data Bases (VLDB)*, pages 613–624, 2005.

[4] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC)*, volume 2910 of *LNCS*, pages 43–58, 2003.

[5] K. Bhattacharya, N. Caswell, S. Kumaran, A. Nigam, and F. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.

[6] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proceedings of 5th International Conference on Business Process Management (BPM)*, Brisbane, Australia, September 2007.

[7] K. Bhattacharya, R. Guttman, K. Lymann, F. F. Heath III, S. Kumaran, P. Nandi, F. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.

[8] K. Bhattacharya, R. Hull, and J. Su. A Data-centric Design Methodology for Business Processes. In J. Cardoso and W. van der Aalst, editors, *Handbook of Research on Business Process Management*. 2009. to appear.

[9] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121:49–107, 2000.

[10] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proc. IJCAI'01*, pages 690–700, 2001.

[11] J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, 1995.

[12] E. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 2000.

[13] O. S. Coalition. OWL-S: Semantic markup for web services, November 2003.

[14] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *Proc. of Intl. Conf. on Database Theory (ICDT)*, 2009.

[15] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[16] M. Fränzle. Verification of hybrid systems. In *Proc. 19th Int. Conf. on Computer Aided Verification (CAV)*, 2007.

[17] C. Gerede, R. Hull, O. Ibarra, and J. Su. Automated composition of e-services: Lookaheads. In *Proc. 2nd Int.*

Conf. on Service-Oriented Computing (ICSOC), 2004.

[18] C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models. In *IEEE International Conference on Service-Oriented Computing and Applications*, 2007.

[19] C. E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *Proceedings of 5th International Conference on Service-Oriented Computing (ICSOC)*, Vienna, Austria, September 2007.

[20] M. L. Ginsberg. Universal planning: an (almost) universally bad idea. *AI Magazine*, 10(4):40–44, 1989.

[21] R. Glushko and T. McGrath. *Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services*. MIT Press, Cambridge, MA, 2005.

[22] S. Grumbach and J. Su. Finitely representable databases. *Journal of Computer & System Sciences*, 55(2):273–298, October 1997.

[23] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, 1969.

[24] R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008*, Monterrey, Mexico, 2008.

[25] R. Hull, F. Llirbat, B. Kumar, G. Zhou, G. Dong, and J. Su. Optimization techniques for data-intensive decision flows. In *Proc. Int. Conf. on Data Engineering*, 2000.

[26] R. Hull, F. Llirbat, E. Simon, J. Su, G. Dong, B. Kumar, and G. Zhou. Declarative workflows that support easy modification and dynamic browsing. In *Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration*, 1999.

[27] J. Jaffar and J. Lassez. Constraint logic programming. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 111–119, 1987.

[28] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer & System Sciences*, 51(1):26–52, 1995.

[29] D. Kozen and C. Yap. Algebraic cell decomposition in NC. In *Proc IEEE Foundations of Computer Science*, pages 515–521, 1985.

[30] G. Kuper, L. Libkin, and J. Paredarns, editors. *Constraint Databases*. Springer Verlag, 2000.

[31] J. Kuster, K. Ryndina, and H. Gall. Generation of BPM for object life cycle compliance. In *Proceedings of 5th International Conference on Business Process Management (BPM)*, 2007.

[32] R. Liu, K. Bhattacharya, and F. Y. Wu. Modeling business contexture and behavior using business artifacts. In *CAiSE*, volume 4495 of *LNCS*, 2007.

[33] M. Moreno and P. Kearney. Integrating AI planning techniques with workflow management system. *Knowledge-Based Systems*, 15(5-6):285–291, 2002.

[34] C. Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3):403–419, 1988.

[35] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Proc. Int. World Wide Web Conf. (WWW)*, 2002.

[36] G. Nelson. A generalization of dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, 1989.

[37] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[38] J. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pages 103–114, 1992.

[39] S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In G. Brewka and J. Lang, editors, *Proceedings of Principles of Knowledge Representation and Reasoning (KR)*, pages 640–650, Sydney, Australia, Sept. 2008.

[40] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In J. McDermott, editor, *Proc. IJCAI'87*, pages 1039–1046, Milan, Italy, 1987. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[41] J. Strosnider, P. Nandi, S. Kumaran, S. Ghosh, and A. Arsanjani. Model-driven synthesis of soa solutions. *IBM Systems Journal*, pages 415–432, 2008.

[42] R. Waldinger. Achieving several goals simultaneously. In E. W. Elcock and D. Michie, editors, *Machine Intelligence*, volume 8, pages 94–136. Wiley, 1977.

[43] J. Wang and A. Kumar. A framework for document-driven workflow systems. In *Business Process Management*, pages 285–301, 2005.

[44] Web Service Modeling Ontology. `http://www.wsmo.org/`.