Workflow Synthesis Based on Instance-Aware Temporal Specification

Vedran Kasalica[1], Natasha Alechina[1], Anna-Lena Lamprecht[1], and Brian Logan[1]

Department of Information and Computing Sciences
Utrecht niversity, 3584 CC Utrecht, Netherlands
{v.kasalica, n.a.alechina, a.l.lamprecht, b.s.logan}@uu.nl

Abstract. A major limitation of many temporal logic-based approaches to automatically synthesising a workflow to accomplish a particular computational task from a set of tools, is their inability to distinguish data instances with the same type signature. This leads to ambiguity in the interpretation of the synthesised solutions, which may prevent the creation of an executable workflow. In this paper, we present a new approach to workflow synthesis that is able to keep track of data instances. In addition, we provide a preliminary extension of the APE (the Automated Pipeline Explorer) framework to support it. We view synthesis as the problem of orchestrating transducers representing tools to achieve a temporal logic specification. We show that the complexity of bounded workflow synthesis (where the number of tools used is known in advance) in our approach is NP-complete. We define dynamic workflow synthesis where the number of times a tool is used is not specified in advance, and show that the dynamic workflow synthesis problem is in PSPACE.

Keywords: workflow synthesis · temporal logic · controller synthesis · computational pipelines · automated workflow composition

1 Introduction

Scientific workflows [8] have become important software artefacts in many scientific disciplines [16]. A scientific workflow is finite directed graph, where the nodes represent operations (performed by available computational tools) and the edges represent data and/or control flow dependencies [17]. In this paper, we focus on computational pipelines, where the graph is acyclic and models only data-flow dependencies. The problem of *workflow synthesis* is, given a specification (what needs to be computed), produce a graph that satisfies the specification.

Program synthesis techniques have been shown to be applicable to the (computational) workflow synthesis problem (see, e.g., [18, 19, 14, 13]). In this approach, a synthesis algorithm is used to construct a computational pipeline using the available tools that satisfies the workflow specification.

In this paper, we focus on temporal logic-based approaches to program synthesis, and in particular on the synthesis approach based on SLTL (Semantic

Linear Time Logic) originally proposed by Steffen et al. [18] and used for the automated composition of scientific workflows in the PROPHETS [15]. The formalism is currently used by the APE [11] framework. SLTL is an extension of Linear Time Logic (LTL) that introduces labeled edges and term taxonomies to enrich the semantics. An SLTL model can be interpreted as a workflow that satisfies a given SLTL formula (temporal goal). The model comprises states representing sets of available data types, and edges representing the operations performed over the data.

A major limitation of this method in practice is its *inability to distinguish data instances*. Scientific workflows frequently reuse already generated data in later stages, and might accumulate multiple different data instances with the same type signature. These have to be distinguishable and separately identifiable to ensure correct data transfer between the components of the workflow. However, in SLTL models, states are collections of available type propositions. As a result, when there are multiple data instances of the same type, their signatures are identical and the framework is not able to distinguish them. This leads to ambiguity in the interpretation of the synthesised solutions, which may prevent the creation of executable workflows.

To illustrate the problem, we use as an example a case study on synthesizing geovisualisation workflows for the generation of maps depicting bird movement patterns in the Netherlands from [10]. The synthesis problem is to generate a workflow over an existing set of GIS tools, that can be used to plot bird movement data and city coordinates on a map. The workflow specification takes as input two csv tables with coordinates (of types "CSV") of bird movements and cities, requires that operations "Plot lines", "Plot points", "Plot coast" and "Plot water" are used, and that an output of type "PostScript" is produced.

However, as the two inputs have the same data signature (csv tables), they cannot be distinguished with this formalism. Thus, the specification cannot ensure that the cities are depicted as dots, and that the bird movement coordinates are connected with lines. Furthermore it cannot even ensure that both input files will be used. Figure 1 shows some possible interpretations of two different SLTL models satisfying the specification. The rectangles represent operations performed, ellipses represent data instances used, and the arrows depict data flows. The arrows also indicate if the data is being used as an input for the operation (red, dotted) or an output of the operation (green, solid). Figures 1(a) and (b) correspond to two different interpretations of the shortest model SLTL-based approaches could provide, with respect to the number of operations performed. However, although the model satisfies the SLTL specification, none of the interpretations uses both of the inputs. Interpretation (a) does not use the birds movement data, while (b) does not use the city coordinates. Figures 1(c) and (d) are interpretations of a "longer" model, which performs two transformation operations, instead of one. Interpretation (c) is indeed a valid solution to the problem. The workflow creates a simple map of the Netherlands, depicting the sea as blue, the coast as green and the bird movements as dots on the map. In contrast, interpretation (d) does not use the bird movement data. Such ambigu-

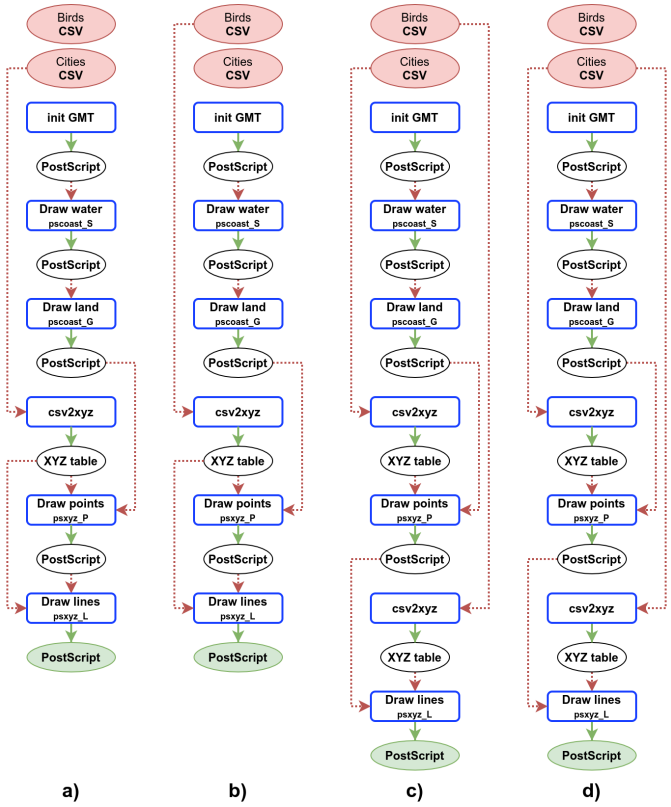

Fig. 1. Possible synthesis solution interpretations.

ities in the language are the main issue of the SLTL-based approach, currently preventing full automation of the workflow composition process.

In this paper, we present a new approach to the synthesis of workflows that preserves and utilises information about data instances in a workflow, and present a preliminary extension of the APE [11] library to support it. Our approach combines and extends workflow synthesis using the temporal logic SLTL and controller synthesis for transducers [7, 1] originally developed for the automated generation of controllers for manufacturing facilities. We show that the complexity of bounded (where the number of times each tool is used is known in advance) workflow synthesis in our approach is NP-complete. We also identify a special case of unbounded synthesis, which we call dynamic workflow synthesis, and show that the dynamic workflow synthesis problem is in PSPACE.

Transition systems with data and some form of quantification in the specification language had been considered, for example, in [4, 2, 5]. Decidability of verification and synthesis in such settings is achieved usually by imposing some kind of boundedness assumption on the domains of states in the transition systems. In this paper, we do not start with the bounded domain assumption, but

boundedness is a consequence of the shape of transition systems corresponding to workflows (that is, that they do not contain loops). Our complexity results are also lower due to the differences in the setting, in particular, a different specification language.

The remainder of the paper is structured as follows. In Section 2 we introduce the formal background (semantically annotated multi-transducers, and an extension of SLTL with first order features, SLTL^x) before describing our new approach to transducer orchestration with temporal goals in Section 3. Then (Section 4) we present a preliminary evaluation of the new approach through application to the aforementioned geovisualization case study. Finally, Section 5 concludes the paper.

2 Transducers and Extended SLTL

In this section we introduce the formal background for a new approach to synthesising workflows using transducers and temporal goals.

2.1 Transducers

We model the tools used in workflow synthesis as semantically annotated multi-transducers. [7]; however there are important differences. A transducer is a finite deterministic automaton with outputs [9]. A multi-transducer [7] has multiple input and output ports (so it can take a tuple of k inputs and produce a tuple of l outputs, for non-negative integers k and l). Semantic annotations on the transitions of a transducer constrain the types of symbols that can be used as inputs, and also specify the types of outputs. We assume that the annotations come from some set of unary predicates L_{types} and that each input and output for each transition can be annotated with zero or finitely many predicates from this set. Transitions of a transducer with k input and l output ports correspond to $k+l$ relations from the set of predicates L_{ops}. To distinguish the input and output arguments in a predicate, we label them with two superscripts, e.g., $P^{k,l}$ corresponds to a predicate of arity $k+l$ where the first k arguments correspond to inputs and the last l to outputs.

Definition 1 (Semantically annotated multi-transducer). *A semantically annotated multi-transducer $T = (\Sigma, S, s_0, f, g, k, l, L_{types}, L_{ops}, Op, Use, Gen)$ is a deterministic transition system with inputs and outputs, where: Σ is the alphabet (of both inputs and outputs), S is a non-empty finite set of states, $s_0 \in S$ is the initial state, $f : S \times \Sigma^k \longrightarrow S$ is the state transition function, $g : S \times \Sigma^k \longrightarrow \Sigma^l$ is the output function, k is the number of T's input ports and l is the number of T's output ports, L_{types} is a finite set of unary predicates, L_{ops} is a finite set of $k+l$-ary predicates, for each transition $t = (s, \mathbf{a}, s', \mathbf{b})$ such that $f(s, \mathbf{a}) = s'$ and $g(s, \mathbf{a}) = \mathbf{b}$, $Op(t) \in L_{ops}$, $Use(t) \in 2^{{L_{types}}^k}$ and $Gen(t) \in 2^{{L_{types}}^l}$.*

We assume that transducers also have a distinguished state s_{err} that takes care of incorrect inputs. We sometimes omit it in the examples below for readability. In this paper, we only use examples of transducers with a single state,

also for brevity. Some tools and resources used in workflows would be more naturally modelled as a multi-state transducer, for example those that admit only a fixed number of queries within a 24 hour period, such as Google Maps API. transducers connected by a port binding introduced below.

We use transducers (representing tools) to generate state transition systems corresponding to a particular instantiation of a workflow. In what follows we essentially treat symbols from Σ as variables for data instances (such as specific files etc.) that are manipulated by the tools represented by the transducers. Only objects (data instances) that satisfy the properties assigned by Use can be used for transitions to a state other than s_{err}, where the corresponding outputs satisfy the properties assigned by Gen. When objects of appropriate type are given as inputs (substituted for the symbols) to a transducer transition, new objects are produced with the properties specified for the output.

The operation can be modeled as $T = (\{tab, ps, plo, err\}, \{s_0, s_{err}\}, s_0, f, g, 2, 1, L_{types}, L_{ops}, Op, Use, Gen)$, where $f(s_0, (tab,ps)) = s_0$ and $g(s_0, (tab,ps)) = plo$; all other transitions lead to s_{err} and output err. The types language $L_{types} = \{XYZ\_table, PostScript\}$. The annotations for the only meaningful transition are: $Op(s_0, (tab, ps), s_0, plo) = psxyz\_P^{2,1}(tab, ps, plo)$, $Use(s_0, (tab, ps), s_0, plo) = \{(XYZ\_table, PostScript)\}$, $Gen((s_0, (tab, ps), s_0, plo) = \{Postscript\}$.

A workflow consists of a number of tools connected together. We model this as a *port binding* of a set of transducers. For a multi-transducer $T^x = (\Sigma, S^x, s_0^x, f^x, g^x, k^x, l^x)$, the input port $1 \leq i < k^x$ is denoted by $in_{x,i}$, and the output port $1 \leq j < l^x$ by $out_{x,j}$. The values at the input port i and output port j of transducer T^x are denoted as $val(in_{x,i})$ and $val(out_{x,j})$, respectively. Similarly, $val(in_x)$ and $val(out_x)$ denote the vectors of values at the input and output ports of T^x. As in [7], we use index $x = 0$ to denote the inputs and outputs of the environment. That is, transducer T^0 specifies the inputs to the workflow and the required outputs: the outputs of the environment are the initial inputs to the set of transducers representing computational tools, $T^1, \ldots, T^m$, and the inputs to the environment are outputs of $T^1, \ldots, T^m$.

Definition 2. *Given a set of transducers $T^0, \ldots, T^m$, a* port binding *c is a set of pairs of the form $(out_{y,j}, in_{x,i})$ (where $x, y \in \{0, \ldots, m\}$ and i, j are port numbers in $\{1, \ldots, k^x\}$ and $\{1, \ldots, l^y\}$, respectively) that represent connections between the output port j of multi-transducer y and input port i of multi-transducer x. A* workflow port binding *in addition satisfies the following constraints:*

- *each input port is connected to at most one output port,*
- *there are no loops, i.e., there is no path along the edges which are either port bindings or links between input and output ports of the same transducer T^x, where $x \in \{1, \ldots, m\}$.*

Given a set of transducers $T^0, \ldots, T^m$, a port binding c and an input tuple $\mathbf{a}$, a sequence of transitions and states $s_0, o_1, \ldots, o_k, s_k$ is generated. The states correspond to sets of ground formulas with unary predicates. The transitions correspond to one step transitions of transducers, that generate new objects

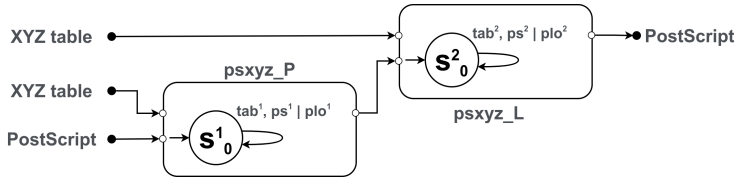

Fig. 2. A workflow modelled as a port binding between two transducers.

with new properties. We assume that type properties of objects do not change (so if e.g., a file object is modified, it becomes a new file).

Figure 2 illustrates a port binding between two transducers, that can plot points (*psxyz_P*) and lines (*psxyz_L*). If the environment is modelled as transducer 0, (*psxyz_P*) as transducer 1 and (*psxyz_L*) as transducer 2, the binding is as follows: $\{(out_{0,1}, in_{2,1}), (out_{0,2}, in_{1,1}), (out_{0,3}, in_{1,2}),$
$(out_{1,1}, in_{2,2}), (out_{2,1}, in_{0,1})\}$.

Given specific input files a, b and c, the binding generates the following state transition system, with states s_0, s_1, s_2, transition between s_0 and s_1 labelled by operation o_1 and transition between s_1 and s_2 labelled by o_2:

$$
\begin{aligned}
s_0 &= \{XYZ\_Table(a), XYZ\_Table(b), PostScript(c)\} \\
o_1 &= \ psxyz\_P^{2,1}(b, c, d) \\
s_1 &= \{XYZ\_Table(a), XYZ\_Table(b), PostScript(c), PostScript(d)\} \\
o_2 &= \ psxyz\_L^{2,1}(a, d, e) \\
s_2 &= \{XYZ\_Table(a), XYZ\_Table(b), PostScript(c), \\
&\qquad PostScript(d), PostScript(e)\}
\end{aligned}
$$

2.2 SLTL^x

In this section, we extend the logic SLTL (Semantic Linear Time Temporal Logic) [18] to be able to talk about objects (data instances). Note that this is not a full first order version of SLTL, in particular, it only has existential quantification, to enable efficient implementation of synthesis. We call the resulting logic SLTL^x.

Similarly to SLTL, SLTL^x presupposes an existence of semantic type hierarchies. The low level operation names and signatures, such as $psxyz\_P^{2,1}(x, y, z)$, are unlikely to be known to the users who specify a workflow. Users are more likely to use a high-level specification of an operation, such as $Plot\_Points^{1,1}(u, v)$ (where u is the input file with coordinates and v is the output map). This necessitates including more operation names in the specification language than those corresponding to the tools, and a representation of a relationship between concrete and abstract operations. In addition, a user may also specify properties of

files which are not in the standard type hierarchy, such as $Birds(a)$ to say that file a contains data on movements of birds.

A new feature of SLTL^x is the introduction of a distinguished binary predicate R to track 'ancestor relation' between objects (an object a is an ancestor of object b, $R(a,b)$, if either $a = b$ or b is an output of an operation that had as one of the inputs an object dependent on a). The reason for needing this relation in the syntax is because while the user may not know the order of operations and the required types of their inputs, they may want to specify that an operation should be performed either directly on the input file or on a file derived from it. For example, a user may require that $Plot\_Points^{1,1}(u,v)$ should be applied either to a file containing coordinates of cities ($Cities(u)$) or to a file that has been obtained from u by performing some processing, $Cities(w) \wedge R(w,u)$.

The syntax of SLTL^x is defined relative to the following alphabet:

- A countable set of variables $Var = \{x, y, z, \ldots\}$,
- A countable set of constants $Con = \{a, b, c, \ldots\}$,
- A finite set L^t of unary predicate symbols that includes L_{types},
- A finite set L^o of predicates symbols that includes L_{ops},
- A distinguished binary predicate R,
- Identity relation between objects, $=$,
- Propositional connectives $true$, $\neg$, $\wedge$,
- Temporal operators $\mathbf{G}$ (always in the future) and $\mathbf{U}$ (until).

Terms are variables or constants. Atomic formulas are of the form $P(t_1, \ldots, t_n)$ where P is an n-ary predicate and $t_1, \ldots, t_n$ are terms, or $t_1 = t_2$.

The set of ground (not containing variables) atomic formulas built using 'concrete' types L_{types} will be denoted by $At^{L_{types}}$. The states will be subsets of $At^{L_{types}}$. The set of ground atoms constructed using 'concrete' operators L_{ops} will be denoted by $At^{L_{ops}}$. Transitions between states correspond to elements of $At^{L_{ops}}$ (we assume that there are no parallel operations by two or more transducers).

We define an 'implements' relation $\triangleright$ between formulas of $At^{L_{ops}}$ and formulas built using L^o to say that a description of a concrete operation is an implementation of an abstract one. This relation is derived from semantic hierarchies for a particular domain. For example, $psxyz\_P^{2,1}(x_1, x_2, x_3)$ implements $Draw\_Points^{1,1}(x_1, x_3)$,
symbolically, $psxyz\_P^{2,1}(x_1, x_2, x_3) \triangleright Draw\_Points^{1,1}(x_1, x_3)$.

The syntax of SLTL^x is given by the following BNF:

$$\Phi \ ::= \ true \mid P(t) \mid R(t_1, t_2) \mid \neg\Phi \mid \Phi \wedge \Phi \mid \langle P(t_1, \ldots, t_n)\rangle\Phi \mid \mathbf{G}\,\Phi \mid \Phi\,\mathbf{U}\,\Phi \mid \exists x\Phi$$

with the restriction that $\exists x$ does not occur negatively in any formula (that is, within the scope of an odd number of $\neg$ symbols).

Given a set of ground atoms A, we define the domain of A, $Dom(A)$, to be the set of all constants occurring in A. Formulas with variables are evaluated in a state s relative to an assignment function $\theta : Var \longrightarrow dom(s)$. For a term t, $[t]_\theta = t$ if t is a constant, and $\theta(t)$ if t is a variable.

Sentences of SLTL^x are formulas with no free (unbound by a quantifier) variables. Workflows are specified by sentences of SLTL^x.

An SLTL^x model $\pi = (s_0, o_1, s_1, o_2, s_2, ..., s_{k-1}, o_k, s_k)$ is a finite alternating sequence of states (subsets of $At^{L_{types}}$) and ground transition relations (elements of $At^{L_{ops}}$). Next we give the truth conditions of SLTL^x formulas on SLTL^x models. Note that given a state s_i in π, it is possible to compute the reflexive and transitive relation R on $dom(s_i)$ from $o_1, \ldots, o_i$.

Definition 3. *Let $\pi = (s_0, o_1, s_1, o_2, s_2, ..., s_{k-1}, o_k, s_k)$ be an SLTL^x model. The relation 'π satisfies formula Φ under assignment θ' ($\pi, \theta \models \Phi$) is as follows:*

$$
\begin{array}{lll}
\pi, \theta \models true & & \\
\pi, \theta \models P(t) & \textit{iff} & P([t]_\theta) \in s_0) \\
\pi, \theta \models t_1 = t_2 & \textit{iff} & [t_1]_\theta = [t_2]_\theta) \\
\pi, \theta \models R(t_1, t_2) & \textit{iff} & R([t_1]_\theta, [t_2]_\theta) \\
\pi, \theta \models \neg\Phi & \textit{iff} & \pi, \theta \not\models \Phi \\
\pi, \theta \models \Phi_1 \wedge \Phi_2 & \textit{iff} & \pi, \theta \models \Phi_1 \wedge \pi, \theta \models \Phi_2 \\
\pi, \theta \models \exists x \Phi & \textit{iff} & \pi, \theta' \models \Phi \textit{ where } \theta' =_x \theta \\
\pi, \theta \models \langle P(t_1, \ldots, t_n) \rangle \Phi & \textit{iff} & (o_1 \triangleright P([t_1]_\theta, \ldots, [t_n]_\theta) \textit{ and } \pi_1, \theta \models \Phi \textit{ and } k > 0 \\
\pi, \theta \models \boldsymbol{G}\Phi & \textit{iff} & \forall i \in \{0, ..., k\} : \pi_i, \theta \models \Phi \\
\pi, \theta \models \Phi_1 \boldsymbol{U} \Phi_2 & \textit{iff} & \exists i \in \{0, ..., k\} : \forall j \in \{0, ..., i-1\} : \\
 & & \pi_j, \theta \models \Phi_1 \textit{ and } \pi_i, \theta \models \Phi_2
\end{array}
$$

where $\theta' =_x \theta$ means that θ' differs from θ at most in its value for x, and π_i is defined as:

$$
\begin{array}{lll}
\pi_i = (s_i, o_i, s_{i+1}, ..., o_k, s_k) & \textit{when} & i \in \{0, ..., k-1\} \\
\pi_i = (s_k) & \textit{when} & i = k
\end{array}
$$

We use the standard definitions for $\vee$ and $\rightarrow$. In addition to the *globally* ($\boldsymbol{G}$) and *until* ($\boldsymbol{U}$) operators as defined above, we will use two additional operators to simplify notation: $\boldsymbol{X}\Phi$, interpreted as $\langle true \rangle \Phi$, denotes the *next-time* operator, and $\boldsymbol{F}\Phi$, interpreted as $true\, \boldsymbol{U}\, \Phi$, denotes *eventually* operator. Note that although we can refer to transitions, the logic is much closer to LTL on finite traces (LTL$_f$) than to Linear Dynamic Logic on finite traces LDL$_f$ in [6].

3 Transducer Synthesis with Temporal Goals

In this section we define two workflow synthesis problems, and analyse their complexity.

Definition 4. *The* bounded workflow synthesis problem *is: given a set of semantically annotated multi-transducers, an SLTL^x formula Φ (the goal formula) and a tuple of inputs, is there a port binding for some subset of transducers such that the resulting SLTL^x model satisfies Φ.*

Theorem 1. *The bounded workflow synthesis problem is NP-complete.*

Proof. For membership in NP, observe that a port binding for a fixed set of transducers and input objects is polynomial in the size of the problem input. Hence it is possible to guess a port binding, generate the corresponding state sequence (which is of finite length polynomial in the input since there are no cycles in the binding), and check whether it satisfies the formula Φ in polynomial time. This means that the problem can be solved by a non-deterministic Turing machines in polynomial time.

For NP hardness, we use a reduction from propositional satisfiability. Let ϕ be a propositional formula over variables $p_1, \ldots, p_n$. The reduction is as follows. The set of $2n$ transducers contains, for each p_i, a transducer that makes p_i true and a transducer that makes p_i false (e.g., p_i can encode type information for some object a). The formula is satisfiable if, and only if, there is a positive answer to the bounded workflow synthesis problem for this set of transducers and a goal formula $\boldsymbol{F}\phi$. □

The bounded version of the problem assumes that the workflow can use at most the explicitly given set of tools (some of which could be copies of the same tool). This is not always the case; for example, it may not be known in advance how many times e.g., a postscript generator will need to be used.

However, it is possible to automatically solve workflow synthesis problems without having to specify the number of copies of each transducer in advance. This can be done by replacing the requirement to produce a fixed port binding by constructing a *dynamic binding*. Intuitively, now the orchestrator is going to construct a new port binding after each transition by the transducers, collect the outputs, and construct a binding again. A useful intuition may be to this of the orchestrator as a planner and of the transducers as operator schemas. The difference from classical planning is as follows: we do not know all the objects in advance, since new objects can be created; properties of objects are not changed once they are established; properties of objects in the goal formula are all unary. We can even specify which operators should be used in the workflow, although without specifying how their arguments relate to other terms in the formula.

Definition 5. *The* dynamic *workflow synthesis problem is as follows: given a finite set of semantically annotated transducers $T^1, \ldots, T^m$, an initial input tuple* $\mathbf{a}$*, a goal formula $\boldsymbol{F}(\phi_1 \wedge \ldots \wedge \phi_v)$ where each ϕ_i is either of the form $\exists x_i \psi(x_i)$ where ψ_i a boolean combination of atoms of the form $P(x_i)$ with $P \in L_{types}$ or of the form $\exists y_1 \ldots y_n \langle P(y_1, \ldots, y_n) \rangle true$, is there a sequence of port bindings such that the initial port binding allocates elements from* $\mathbf{a}$ *to some input ports of $T^1, \ldots, T^m$, and each subsequent binding allocates outputs from the previous step to (possibly different) input ports of $T^1, \ldots, T^m$, and the transition system generated by the transducers under these bindings satisfies the goal formula.*

Theorem 2. *The dynamic workflow synthesis problem is PSPACE-complete.*

Proof. We introduce some notation and terminology first. Let us denote by $K = \Sigma_{j=1,\ldots,m} k^m$ the maximal number of input ports that can be used simultaneously in parallel.

Observe that the set of types L_{types} is finite; a complete description of an object in terms of the types in L_{types} is a conjunction of atoms and negated atoms for each type $P \in L_{types}$. There are $2^{|L_{types}|}$ such complete descriptions, which we will refer to as supertypes.

Let us consider first the case where the goal formula is of the form $\boldsymbol{F}\exists x\psi(x)$. If an object satisfying ψ can be constructed at all, there is a sequence of states leading to a state which contains an object satisfying ψ. This sequence does not have repetitions. Each state can be uniquely described as an allocation of one of $2^{|L_{types}|}$ supertypes to each of possible K input ports (the outputs are produced deterministically), so there are $2^{|L_{types}|\times K}$ different states. Clearly, the sequence leading from initial state to a $\psi(x)$ state can be exponentially long. However, similarly to classical planning, a state can be represented in polynomial space by listing at most $|L_{types}|$ positive properties for each of K input ports. A path-exists(s_1, s_2, N) algorithm that checks the existence of a path of length N between states s_1 and s_2 (where s_2 satisfies the goal test, that is outputs a $\psi(x)$ object) works in polynomial space; for $N = 1$ it checks whether $s_1 = s_2$ or there is a single step transition between them, for $N > 1$ it recursively calls plan-exists$(s_1, s_3, \lceil N/2 \rceil)$ and plan-exists$(s_3, s_2, \lfloor N/2 \rfloor)$; note that N represented in binary takes $O(logN)$ space, so it polynomial in the input size [3].

In order to check whether the required operators have been used, we can modify the path-exists(s_1, s_2, N) algorithm to return the set of operator names encountered on the path from s_1 to s_2. Observe that this set of names (unlike the complete list of all ground operator formulas on the path) is polynomial in the input size.

The problem of generating several objects with specified types is no harder than for a single object, because properties of objects persist.

PSPACE-hardness is by straightforward reduction from STRIPS planning. □

4 Evaluation

The preliminary evaluation of the new approach in practice compares it to the SLTL synthesis approach that is currently implemented in the APE [11] framework (as the latest implementation of that formalism).

In order to be able to reuse the existing domain annotations, the new implementation extends the existing infrastructure of the APE framework. It addresses the bounded workflow synthesis problem. Although the bounded synthesis has its limitations, an iterative deepening approach demonstrated to be effective in practice in the previous APE evaluations.

In the evaluation we focus on the main difference between the approaches, namely the ability of the new approach to distinguish data instances with the same signature. In our case those are the two input files, containing bird movement and city coordinates. With the extended specification logic SLTL^x we can now describe the problem better by restricting the usage of the inputs, as follows:

$$CSV(a) \wedge CSV(b) \wedge Cities(a) \wedge Birds(b) \wedge \exists a_1 \exists b_1 \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5$$
$$(\boldsymbol{F}(R(b, b_1) \wedge \langle Plot\_Lines^{1,1}(b_1, x_1)\rangle true) \wedge \boldsymbol{F}(R(a, a_1) \wedge$$
$$\langle Plot\_Points^{1,1}(a_1, x_2)\rangle true) \wedge \boldsymbol{F}\langle Plot\_Coast^{0,1}(x_3)\rangle true \wedge$$
$$\boldsymbol{F}\langle Plot\_Water^{0,1}(x_4)\rangle true \wedge \boldsymbol{F}(PostScript(x_5) \wedge R(x_1, x_5) \wedge R(x_2, x_5)$$
$$\wedge R(x_3, x_5) \wedge R(x_4, x_5) \wedge \neg \boldsymbol{X} true))$$

This encoding ensures that the bird movements will be labeled "Birds" and transformed (if needed) to be connected by lines. Similarly, the cities will be labeled "Cities" and depicted as dots.

The new approach eliminates the ambiguity and allows more concrete problem specification. Workflows in Figures 1(a)-(d) will be distinguished and only those satisfying the specification will be synthesised in the process.

The new implementation synthesis resulted in 16 workflows of length 7, where the length corresponds to the number of operations needed to be performed. Each of the suggested workflows represents a permutation of the desired workflow presented in Figure 1(c), it satisfies our specification, and can be used to automatically generate the desired map. Furthermore, the undesired permutations can be omitted by restricting the order of operations in the problem specification.

Although the current implementation does not yet support the full expressiveness of the new approach, we are currently working on extending APE to support it. In order to simplify the encoding of the new logical structures, the new implementation will not keep the CNF encoding of the specification. Instead it will transform it into SMT-LIB 2.0 format and use an off-the-shelf SMT solver to find solutions. The evaluation of such implementation should then focus on more complex use cases, such as synthesis in proteomics domain [12].

5 Conclusion

The creation of scientific workflows can be challenging. Workflow developers need to identify the relevant workflow components from potentially huge collections of computational tools, and compose them correctly (order, type compatibility) to solve a given computational problem. Automating workflow synthesis reduces the time required and the likelihood of errors. Here we present a new approach for the synthesis of computational workflows using transducers and temporal goals. It keeps track of the data instances in the synthesised workflow and thus overcomes a major limitation of the original temporal logic-based approach. We defined a bounded and a dynamic variant of the workflow synthesis problem, and proved that they are NP-complete and in PSPACE, respectively. To evaluate the new approach, we extended the existing workflow synthesis framework APE with a first implementation of it. Application to an existing case study confirmed that the added instance-awareness improves the specificity of the synthesized workflows, while not notably impacting synthesis execution time.

References

1. Alechina, N., Brázdil, T., De Giacomo, G., Felli, P., Logan, B., Vardi, M.Y.: Unbounded orchestrations of transducers for manufacturing. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 2646–2653 (2019)
2. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of agent-based artifact systems. J. Artif. Intell. Res. **51**, 333–376 (2014). https://doi.org/10.1613/jair.4424
3. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence **69**(1-2), 165–204 (1994)
4. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and synthesis in description logic based dynamic systems. In: Faber, W., Lembo, D. (eds.) Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Proceedings. Lecture Notes in Computer Science, vol. 7994, pp. 50–64. Springer (2013)
5. De Giacomo, G., Lespérance, Y., Patrizi, F.: Bounded situation calculus action theories. Artif. Intell. **237**, 172–203 (2016)
6. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. In: Proceedings of IJCAI 2015. pp. 1558–1564. AAAI Press (2015)
7. De Giacomo, G., Vardi, M.Y., Felli, P., Alechina, N., Logan, B.: Synthesis of orchestrations of transducers for manufacturing. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
8. Garijo, D.: AI Buzzwords Explained: Scientific Workflows. AI Matters **3**(1), 4–8 (May 2017). https://doi.org/10.1145/3054837.3054839
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 2nd edition. ACM SIGACT News **32**(1), 60–65 (2001)
10. Kasalica, V., Lamprecht, A.L.: Workflow Discovery Through Semantic Constraints: A Geovisualization Case Study. In: Computational Science and Its Applications – ICCSA 2019. pp. 473–488. Springer International Publishing, Cham (2019)
11. Kasalica, V., Lamprecht, A.L.: APE: A Command-Line Tool and API for Automated Workflow Composition. In: Krzhizhanovskaya, V.V., Závodszky, G., Lees, M.H., Dongarra, J.J., Sloot, P.M.A., Brissos, S., Teixeira, J. (eds.) Computational Science – ICCS 2020. pp. 464–476. Springer International Publishing, Cham (2020)
12. Kasalica, V., Schwämmle, V., Palmblad, M., Ison, J., Lamprecht, A.L.: Ape in the wild: Automated exploration of proteomics workflows in the bio. tools registry. Journal of proteome research **20**(4), 2157–2165 (2021)
13. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-Based Loose Programming. In: Proc. of the QUATIC 2010, Portugal. pp. 262–267. IEEE (2010)
14. Lustig, Y., Vardi, M.Y.: Synthesis from Component Libraries. In: de Alfaro, L. (ed.) Foundations of Software Science and Computational Structures. pp. 395–409. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2009)
15. Naujokat, S., Lamprecht, A.L., Steffen, B.: Loose Programming with PROPHETS. In: Proc. of FASE 2012, Estonia. LNCS, vol. 7212, pp. 94–98 (2012)
16. Perkel, J.M.: That's the way we flow. Computational pipelines turn raw data into reproducible scientific knowledge. Nature **573**(7772), 149–150 (Sep 2019)
17. Qin, J., Fahringer, T.: Scientific Workflows: Programming, Optimization, and Synthesis with ASKALON and AWDL. Springer-Verlag, Berlin Heidelberg (2012)
18. Steffen, B., Margaria, T., Freitag, B.: Module Configuration by Minimal Model Construction. Tech. rep., Fakultät für Mathematik und Informatik, Universität Passau (1993)
19. Van Der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Web Services and Formal Methods. pp. 1–23. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2006)