Data Centric Workflows for Crowdsourcing

Pierre Bourhis¹, Loïc Héluët^{2(✉)}, Zoltan Miklos³, and Rituraj Singh³

¹ CNRS, Univ. Lille, Lille, France

`pierre.bourhis@univ-lille.fr`

² INRIA Rennes, Rennes, France

`loic.helouet@inria.fr`

³ Univ. Rennes, Rennes, France

`{zoltan.miklos,rituraj.singh}@irisa.fr`

Abstract. Crowdsourcing consists in hiring workers on internet to perform large amounts of simple, independent and replicated work units, before assembling the returned results. A challenge to solve intricate problems is to define orchestrations of tasks, and allow higher-order answers where workers can suggest *a process to obtain data* rather than a *plain answer*. Another challenge is to guarantee that an orchestration with correct input data terminates, and produces correct output data. This work proposes *complex workflows*, a data-centric model for crowdsourcing based on orchestration of concurrent tasks and higher order schemes. We consider termination (whether some/all runs of a complex workflow terminate) and correctness (whether some/all runs of a workflow terminate with data satisfying FO requirements). We show that existential termination/correctness are undecidable in general excepted for specifications with bounded recursion. However, universal termination/correctness are decidable when constraints on inputs are specified in a decidable fragment of FO, and are at least in *co-2EXPTIME*.

1 Introduction

Crowdsourcing leverages intelligence of crowd to realize tasks where human skills still outperform machines [26]. It was successful in contributive science initiatives, such as CRUK's Trailblazer [3], Galaxy Zoo [5], etc. Most often, a crowdsourcing project consists in deploying a huge number of tasks that can be handled by humans in a reasonable amount of time. Generally, work units are simple micro-tasks, that are independent, cheap, repetitive, and take a few minutes to an hour to complete. They can be labeling of images, writing scientific blogs, etc. The requester publishes the tasks on a platform with a small incentive (a few cents, reputation gain, goodies ...), and waits for the participation from the crowd. However, many projects, and in particular scientific workflows, can be seen as a coordination of high-level composite tasks. As noted by [39], composite tasks are not or poorly supported by crowdsourcing platforms. Crowdsourcing markets such as Amazon Mechanical Turk [1], Foule Factory [4], CrowdFlower

[2], etc. already propose interfaces and APIs to access crowds, but the specification of crowd based complex processes is still in its infancy. Some works propose solutions for data acquisition and management or deployment of *workflows*, mainly at the level of micro-tasks [18,30]. Crowdforge uses Map-Reduce techniques to solve complex tasks [24]. Turkit [31] builds on an imperative language embedding calls to services of a crowdsourcing platform, which requires programming skills to design complex orchestrations. Turkomatic [27] and [42] implement a Price, Divide and Solve (PDS) approach: crowd workers divide tasks into orchestrations of subtasks, up to the level of micro-tasks. However, current PDS approaches ask clients to monitor workflows executions. In this setting, clients cannot use PDS crowdsourcing solutions as high-level services.

The next stage of crowdsourcing is hence to design more involved processes still relying on the crowd. A first challenge is to fill the gap between a high-level process that a requester wants to realize, and its implementation in terms of micro-tasks composition. Moving from one description level to the other is not easy, and we advocate the use of expertise of the crowd for such refinement. This can be achieved with higher-order answers, allowing a knowledgeable worker to return an orchestration of simpler tasks instead of a crisp answer to a question. A second challenge in this setting is to give guarantees on the termination (whether some/all runs terminate) and correctness of the overall process (whether the output data returned by a crowdsourced process meet some requirements w.r.t input data).

This paper proposes a data-centric model called *complex workflows*, that define orchestrations of tasks with higher-order schemes, and correctness requirements on input and output data of the overall processes. Complex workflows are mainly orchestrations of data transformation tasks, that start from initial input datasets that are explicitly given as crisp data or represented symbolically with a logical formula. Workers can contribute to micro-tasks (input data, add tags ...) or refine tasks at runtime. Some easy tasks (simple SQL queries or record updates) are automated. Input/Output (I/O) requirements are specified with a fragment of FO. We address the question of termination (whether all/some runs terminate) and correctness (whether all/some runs terminate and meet the I/O requirements). Due to higher-order, complex workflows are Turing complete, and hence existence of a terminating run is undecidable. However, one can decide whether **all** runs of a complex workflow terminate as soon as initial data is fixed or specified in an FO fragment where satisfiability is decidable. Existential termination becomes decidable with the same constraints on initial data as soon as recursion in higher-order rewritings is bounded. We then show that existential correctness is undecidable, and universal correctness is decidable if constraints on data are expressed in a decidable fragment of FO. The complexity of termination and correctness depends mainly on the length of runs, and on the complexity of satisfiability for the FO fragment used to specify initial data and I/O requirements. It is at least in (co-)2EXPTIME for the simplest fragments of FO, but can increase to an n -fold complexity for FO fragments that use both existential and universal quantification.

Several formal models have been proposed for orchestration of tasks and verification in data-centric systems or business processes. Workflow nets [40, 41], a variant of Petri nets, address the control part of business processes, but data is not central. Orchestration models such as ORC [23] or BPEL [37] can define business processes, but cannot handle dynamic orchestrations and are not made to reason on data. [10] proposes a verification scheme for a workflow model depicting business processes with external calls, but addresses operational semantics issues (whether some “call pattern” can occur) rather than correctness of the computed data. Several variants of Petri nets with data have been proposed. Obviously, colored Petri nets [21] have a sufficient expressive power to model complex workflows. But they are defined at a low detail level, and encoding higher-order necessarily have to be done through complex color manipulations. Complex workflows separate data, orchestration, and data transformations. Several variants of PNs where transitions manipulate global variables exist (see for instance [15]), but they cannot model evolution of unbounded datasets. Other variants of PNs with tokens that carry data have also been proposed in [28]. Termination and boundedness are decidable for this model, but with non-elementary complexity, even when equality is the only predicate used. The closest Petri net variant to complex workflows are Structured Data nets [9], in which tokens carry structured data. Higher order was introduced in *nested* Petri nets [32] a model where places contain sub-nets. Interestingly, termination of nested nets is decidable, showing that higher-order does not imply Turing completeness. Data-centric models and their correctness have also been considered. Guarded Active XML [7] (GAXML) is a model where guarded services are embedded in structured data. Though GAXML does not address explicitly crowdsourcing nor higher-order, the rewritings performed during service calls can be seen as a form of *task* refinement. Restrictions on recursion in GAXML allows for verification of Tree LTL (a variant of LTL where propositions are statements on data). More recently, [8] has proposed a model for collaborative workflows where peers have a local view of a global instance, and collaborate via local updates. With some restrictions, PLTL-FO (LTL-FO with past operators) is decidable. Business artifacts were originally developed by IBM [36], and verification mechanisms for LTL-FO (LTL formula embedding FO statements over universally quantified variables) were proposed in [14, 25]. LTL-FO is decidable for subclasses of artifacts with data dependencies and arithmetic. [16] considers systems composed of peers that communicate asynchronously. LTL-FO is decidable for systems with bounded queues. Business artifacts allow for data inputs during the lifetime of an artifact, but are static orchestrations of guarded tasks, described as legal relations on datasets before and after execution of a task. Further, LTL-FO verification focuses mainly on dynamics of systems (termination, reachability) but does not address correctness. [19] considers data-centric dynamic systems (DCDS), i.e. relational databases equipped with guarded actions that can modify their contents, and call external services. Fragments of FO μ -calculus are decidable for DCDS with bounded recursion.

Fig. 1. a) A simple actor popularity poll, and b) a rule example.

2 Motivation

Our objectives are to provide tools to develop applications in which human actors execute tasks or propose solutions to realize a complex task, and check the correctness of the designed services. The envisioned scenario is the following: a client provides a coarse grain workflow depicting important phases of a complex task to process data, and a description of the expected output. The tasks can be completed in several ways, but cannot be fully automated. It is up to a pool of crowdworkers to complete them, possibly after refining difficult tasks up to the level of an orchestration of easy or automated micro-tasks.

The crowdsourcing model presented in Sect. 4 is currently used to design a real example, the SPIPOLL initiative [6]. The objective of SPIPOLL is to study populations of pollinating insects. The standard processes used by SPIPOLL call for experience of large pools of workers to collect, sort and tag large databases containing insects pictures. The features provided by our model fit particularly with the needs of SPIPOLL, as tagging pictures of rare insects is not a simple annotation task, and usually calls for complex validation processes.

Let us illustrate the needs and main features of complex workflows, refinement, and human interactions on a simpler example. A client (for instance a newspaper) wants to rank the most popular actors of the moment, in the categories *comedy*, *drama* and *action* movies. The task is sent to a crowdsourcing platform as a high-level process decomposed into three sequential phases: first a collection of the most popular actors, then a selection of the 50 most cited names, followed by a classification of these actors in comedy/drama/action category. The ranking ends with a vote for each category, that asks contributors to associate a score to each name. The client does not input data to the system, but has some requirements on the output: it is a dataset D_{ranked} with relational schema $r_{out}(name, cites, category, score)$, where *name* is a key, *cites* is an integer that gives the number of cites of an actor, *category* ranges over $\{drama, comedy, action\}$ and *score* is a rational number between 0 and 10. Further, every actor appearing in the final database should have a *score* and a number of *cites* greater than 0. Obviously, there are several ways to collect actors names, several ways to associate a category tag, to vote, etc. However, the clients needs are defined in terms of an orchestration of high-level tasks,

without information on how the crowd will complete them, and without any termination guarantee. The orchestration is depicted in Fig. 1-a. It starts from an empty input dataset $D_\emptyset$, and returns an actor popularity dataset D_{ranked} .

In the rest of the paper, we formalize complex workflows and their semantics (Sect. 4). We then address termination (Sect. 5) and correctness (Sect. 6). More precisely, given a complex workflow CW , we consider the following problems:

Universal Termination: Does every run of CW terminate?

Existential Termination: Is there an input for which at least one run of CW terminates?

Universal Correctness: For a given I/O requirement $\psi^{in,out}$ relating input and output data, does every run of CW terminate and satisfy $\psi^{in,out}$?

Existential Correctness: For a given I/O requirement $\psi^{in,out}$, is there a particular input and at least one run of CW that terminates and satisfies $\psi^{in,out}$?

3 Preliminaries

We use a standard relational model [13], i.e., data is organized in *datasets*, that follow *relational schemas*. We assume finite set of domains $\mathbf{dom} = dom_1, \dots, dom_s$, a finite set of attribute names $\mathbf{att}$ and a finite set of relation names $\mathbf{relnames}$. Each attribute $a_i \in \mathbf{att}$ is associated with a domain $dom(a_i) \in \mathbf{dom}$. A *relational schema* (or table) is a pair $rs = (rn, A)$, where rn is a relation name and $A \subseteq \mathbf{att}$ denotes a finite set of attributes. Intuitively, attributes in A are column names in a table, and rn the table name. The *arity* of rs is the size of its attributes set. A *record* of a relational schema $rs = (rn, A)$ is tuple $rn(v_1, \dots, v_{|A|})$ where $v_i \in dom(a_i)$ (it is a row of the table), and a *dataset* with relational schema rs is a multiset of records of rs . A *database schema* DB is a non-empty finite set of tables, and an instance over a database DB maps each table in DB to a dataset.

We address properties of datasets $D_1, \dots, D_k$ with *First Order logic* (FO) and predicates on records. For simplicity and efficiency, we will consider predicates defined as sets of linear inequalities over reals and integer variables. We use FO to define relations between the inputs and outputs of a task, and requirements on possible values of inputs and outputs of a complex workflow.

Definition 1 (First Order). A First Order formula (in prenex normal form) over a set of variables $\vec{X} = x_1, \dots, x_n$ is a formula of the form $\varphi ::= \alpha(\vec{X}).\psi(\vec{X})$ where $\alpha(\vec{X})$ is an alternation of quantifiers and variables of $\vec{X}$, i.e. sentences of the form $\forall x_1 \exists x_2, \dots$ called the prefix of φ and $\psi(\vec{X})$ is a quantifier free formula called the matrix of φ . $\psi(\vec{X})$ is a boolean combinations of atoms of the form $rn_i(x_1, \dots, x_k) \in D_i$, $P_j(x_1, \dots, x_n)$, where $rn_i(x_1, \dots, x_k)$'s are relational statements, and $P_j(x_1, \dots, x_n)$'s are boolean function on $x_1, \dots, x_n$ called predicates.

In the rest of the paper, we consider variables that either have finite domains or real valued domains, and predicates specified by simple linear inequalities of dimension 2, and in particular equality of variables, i.e. statements of the form $x_i = x_j$. One can decide in NP if a set of linear inequalities has a valid assignment, and in polynomial time [22] if variables values are reals. Let $\vec{X}_1 = \{x_1, \dots, x_k\} \subseteq \vec{X}$. We write $\forall \vec{X}_1$ instead of $\forall x_1. \forall x_2 \dots \forall x_k$, and $\exists \vec{X}_1$ instead of $\exists x_1. \exists x_2 \dots \exists x_k$. We use w.l.o.g. formulas of the form $\forall \vec{X}_1 \exists \vec{X}_2 \dots \psi(X)$ or $\exists \vec{X}_1 \forall \vec{X}_2 \dots \psi(X)$, where $\psi(X)$ is a quantifier free matrix, and for every $i \neq j$, $\vec{X}_i \cap \vec{X}_j = \emptyset$. Every set of variables $\vec{X}_i$ is called a *block*. By allowing blocks of arbitrary size, and in particular empty blocks, this notation captures all FO formulas in prenex normal form. We denote by $\varphi_{[t_1/t_2]}$ the formula obtained by replacing every instance of term t_1 in φ by term t_2 . Each variable x_i in $\vec{X}$ has a domain $Dom(x_i)$. A variable assignment is a function μ that associates a value d_x from $Dom(x)$ to each variable $x \in \vec{X}$.

Definition 2 (Satisfiability). *A variable free formula is satisfiable iff it evaluates to true. A formula of the form $\exists x, \phi$ is satisfiable iff there exists a value $d_x \in Dom(x)$ such that $\phi_{[x/d_x]}$ is satisfiable. A formula of the form $\forall x, \phi$ is satisfiable iff, for every value $d_x \in Dom(x)$, $\phi_{[x/d_x]}$ is satisfiable.*

It is well known that satisfiability of an FO formula is undecidable in general, but it is decidable for several fragments. The *universal fragment* (resp. *existential fragment*) of FO is the set of formulas of the form $\forall \vec{X}. \varphi$ (resp. $\exists \vec{Y}. \varphi$) where φ is quantifier free. We denote by $\forall FO$ the universal fragment of FO and by $\exists FO$ the existential fragment. Checking satisfiability of the existential/universal fragment of FO can be done non-deterministically in polynomial time. In our setting, where atoms in FO formula are relational statements and linear inequalities, satisfiability of formulas with only universal or existential quantifiers is decidable and (co)-NP-complete. We refer interested readers to [12] for a proof.

One needs not restrict to existential or universal fragments of FO to get decidability of satisfiability. A well known decidable fragment is (FO^2) , that uses only two variables [35]. However, this fragment forbids atoms of arity greater than 2, which is a severe limitation when addressing properties of datasets. The *BS* fragment of FO is the set of formulas of the form $\exists \vec{Y}_1. \forall \vec{X}_2. \psi$, where ψ is quantifier free, may contain predicates, but no equality. The *Bernays-Schonfinkel-Ramsey* fragment of FO [11] (BSR-FO for short) extends the *BS* fragment by allowing equalities in the matrix ψ . Satisfiability of a formula in the BS or BSR fragment of FO is NEXPTIME-complete (w.r.t. the size of the formula) [29].

Recent results [38] exhibited a new fragment, called the *separated fragment* of FO, defined as follows: Let $Vars(A)$ be the set of variables appearing in an atom A . We say that two sets of variables $Y, Z \subseteq X$ are separated in a quantifier free formula $\phi(X)$ iff for every atom At of $\phi(X)$, $Vars(At) \cap Y = \emptyset$ or $Vars(At) \cap Z = \emptyset$. A formula in the *Separated Fragment* of FO (*SF-FO* for short)

is a formula of the form $\forall \vec{X}_1. \exists \vec{Y}_2. \dots \forall \vec{X}_n. \exists \vec{Y}_n \phi$, where $\vec{X}_1 \dots \cup \vec{X}_n$ and $\vec{Y}_1 \dots \cup \vec{Y}_n$ are separated. The *SF* fragment is powerful and subsumes the *Monadic Fragment* [33] (where predicates can only be unary) and the BSR fragment. Every separated formula can be rewritten into an equivalent BSR formula (which yields decidability of satisfiability for SF formulas), but at the cost of an n -fold exponential blowup in the size of the original formula. Satisfiability of a separated formula φ is hence decidable [38], but with a complexity in $O(2^{\downarrow n|\varphi|})$.

A recent extension of FO^2 called FO^2BD allows atoms of arbitrary arity, but only formulas over sets of variables where at most two variables have unbounded domain. Interestingly, FO^2BD formulas are closed under computation of weakest preconditions for a set of simple SQL operations [20]. We show in Sect. 4 that conditions needed for a non-terminating execution of a complex workflow are in $\forall FO$, and that $\forall FO$, $\exists FO$, BSR-FO, SF-FO are closed under precondition calculus.

4 Complex Workflows

This section formalizes the notion of *complex workflow*, and gives their semantics through operational rules. This model is inspired by artifacts systems [14], but uses higher-order constructs (task decomposition), and relies on human actors (the *crowdworkers*) to complete tasks. We assume a *client* willing to use the power of crowdsourcing to realize a *complex task* that needs human contribution to collect, annotate, or organize data. This complex task is specified as a workflow that orchestrates elementary tasks or other complex tasks. The client can input data to the system (i.e. enter a dataset D_{in}), and may have a priori knowledge on the relation between the contents of his input and the plausible outputs returned after completion of the complex workflow. This scenario fits several types of applications such as opinion polls, citizen participation, etc. High-level answers of workers are seen as workflow refinements, and elementary tasks realizations are simple operations that transform data. During an execution of a complex workflow, we consider that each worker is engaged in the execution of at most one task.

Tasks: A task t is a work unit designed to transform input data into output data. It can be a high-level description submitted by a client, a very basic atomic task that can be easily realized by a single worker (e.g. tagging images), a task that can be fully automated, or a complex task that requires an orchestration of sub-tasks to reach its objective. We define a set of tasks $\mathcal{T} = \mathcal{T}_{ac} \uplus \mathcal{T}_{cx} \uplus \mathcal{T}_{aut}$ where $\mathcal{T}_{ac}$ is a set of *atomic tasks* that can be completed in one step by a worker, $\mathcal{T}_{cx}$ is a set of *complex tasks* which need to be refined as an orchestration of smaller sub-tasks to produce an output, and $\mathcal{T}_{aut}$ is a set of automated tasks performed by a machine (e.g. simple SQL queries: selection, union, projection ...). Tasks in $\mathcal{T}_{ac}$ and $\mathcal{T}_{aut}$ cannot be refined, and tasks in $\mathcal{T}_{aut}$ do not require contribution of a worker.

Definition 3 (Workflow). A workflow is a labeled acyclic graph $W = (N, \rightarrow, \lambda)$ where N is a finite set of nodes, modeling occurrences of tasks, $\rightarrow \subseteq N \times N$ is a precedence relation, and $\lambda: N \rightarrow \mathcal{T}$ associates a task name to each node. A node $n \in N$ is a source iff it has no predecessor, and a sink iff it has no successor.

In the rest of the paper, we fix a finite set of tasks $\mathcal{T}$, and denote by $\mathcal{W}$ the set of all possible workflows over $\mathcal{T}$. Intuitively, if $(n_1, n_2) \in \rightarrow$, then an occurrence of task named $\lambda(n_1)$ represented by n_1 must be completed before an occurrence of task named $\lambda(n_2)$ represented by n_2 , and the data computed by n_1 is used as an input for n_2 . We denote by $\min(W)$ the set of sources of W , by $\text{succ}(n_i)$ the set of successors of a node n_i , and by $\text{pred}(n_i)$ its predecessors. The size of W is the number of nodes in N and is denoted $|W|$. We assume that when a task in a workflow has several predecessors, its role is to aggregate data provided by preceding tasks, and when a task has several successors, its role is to distribute excerpts from its input datasets to its successors. With this convention, one can model situations where a large database is split into smaller datasets of reasonable sizes that are then processed independently. We denote by $W \setminus \{n_i\}$ the restriction of W to $N \setminus \{n_i\}$, i.e. a workflow from which node n_i is removed along with all edges which origins or goals are node n_i . We assume some well-formedness properties of workflows: Every workflow has a single sink node n_f . Informally, we can think of n_f as the task that returns the dataset computed during the execution of the workflow. There exists a path from every node n_i of W to the sink n_f . The property prevents from launching tasks which results are never used to build an answer to a client. We define a *higher-order* answer as a *refinement* of a node n in a workflow by another workflow. Intuitively, n is simply replaced by W' in W .

Definition 4 (Refinement). Let $W = (N, \rightarrow, \lambda)$ be a workflow, $W' = (N', \rightarrow', \lambda')$ be a workflow with a unique source node $n'_{src} = \min(W')$, a unique sink node n'_f such that $N \cap N' = \emptyset$. The refinement of $n \in N$ by W' in W is the workflow $W_{[n/W']} = (N_{[n/W]}, \rightarrow_{[n/W]}, \lambda_{[n/W]})$, where $N_{[n/W]} = (N \setminus \{n\}) \cup N'$, $\lambda_{[n/W]}(n) = \lambda(n)$ if $n \in N$, $\lambda'(n)$ otherwise, and $\rightarrow_{[n/W]} = \rightarrow \cup \{(n_1, n_2) \in \rightarrow \mid n_1 \neq n \wedge n_2 \neq n\} \cup \{(n_1, n'_{src}) \mid (n_1, n) \in \rightarrow\} \cup \{(n'_f, n_2) \mid (n, n_2) \in \rightarrow\}$.

Definition 5. A Complex Workflow is a tuple $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ where $\mathcal{T}$ is a set of tasks, $\mathcal{U}$ a finite set of workers, $\mathcal{R} \subseteq \mathcal{T} \times 2^{\mathcal{W}}$ is a set of rewriting rules with $\mathcal{W}$ a finite set of workflows, and $sk \subseteq (\mathcal{U} \times \mathcal{R}) \cup (\mathcal{U} \times \mathcal{T}_{ac})$ defines workers competences. W_0 is an initial workflow, that contains a single source node n_{init} and a single sink node n_f .

We assume that in every rule $(t, W) \in \mathcal{R}$, the labeling λ of W is injective. This results in no loss of generality, as one can create copies of a task for each node in W , but simplifies proofs and notations afterwards. Further, W has a unique source node $\text{src}(W)$. The relation sk specifies which workers have the right to perform or refine a particular task. This encodes a very simple competence model. We refer to [34] for further studies on competences and more elaborated competence models.

Fig. 2. A refinement of node n_1 using rule (t_{like}, W_{like}) of Fig. 1-b.

Let us consider the example of Fig. 2-left: a workflow contains a complex task t_{like} which objective is to rank large collections of images of different animals with a score between 0 and 10. The relational schema for the dataset D used as input for t_{like} is a collection of records of the form $Picdata(nb, name, kind)$ where nb is a key, $name$ is an identifier for a picture, $kind$ denotes the species obtained from former annotation of data by crowdworkers. Let us assume that a worker u knows how to handle task t_{like} (i.e. $(u, t_{like}) \in sk$), and wants to divide dataset D into three disjoint datasets containing pictures of cats, dogs, and other animals, rank them separately, and aggregate the results. This is captured by the rule $R = (t_{like}, W_{like})$ of Fig. 1-b, where node n_0 is an occurrence of an automated tasks that splits an input dataset into datasets containing pictures of dogs, cats, and other animals, n_1, n_2, n_3 represent tagging tasks for the respective animal kinds, and node n_f is an automated task that aggregates the results obtained after realization of preceding tasks. A higher-order answer of worker u is to apply rule R to refine node n_1 in the original workflow with W_{like} . The result is shown in Fig. 2-right.

It remains to show how automated and simple tasks are realized and process their input data to produce output data. At every node n representing a task $t = \lambda(n)$, the relational schemas of all input (resp. output) datasets are known and denoted $rs_1^{in}, \dots, rs_k^{in}$ (resp. $rs_1^{out}, \dots, rs_k^{out}$). We denote by $\mathcal{D}^{in} = D_1^{in}, \dots, D_k^{in}$ the set of datasets provided by predecessors of t , and by $\mathcal{D}^{out} = D_1^{out}, \dots, D_q^{out}$ the set of output datasets computed by task t . During a run of a complex workflow, we allow tasks executions **only for nodes which inputs are not empty**. The contents of every D_i^{out} is the result of one of the operations below:

SQL-LIKE OPERATIONS: We allow standard SQL operations:

- **Selection:** For a given input dataset D_i^{in} with schema $rn(x_1, \dots, x_n)$ and a predicate $P(x_1, \dots, x_n)$, compute $D_j^{out} = \{rn(x_1, \dots, x_n) \mid rn(x_1, \dots, x_n) \in D_i^{in} \wedge P(x_1, \dots, x_n)\}$.
- **Projection:** For a given input dataset D_i^{in} with schema $rn(x_1, \dots, x_n)$ and an index $k \in 1..n$ compute $D_j^{out} = \{rn(x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n) \mid rn(x_1, \dots, x_n) \in D_i^{in}\}$.
- **Insertion/deletion:** For a given input dataset D_i^{in} and a fixed tuple $rn(a_1, \dots, a_n)$, compute $D_j^{out} = D_i^{in} \cup \{rn(a_1, \dots, a_n)\}$ (resp. $D_j^{out} = D_i^{in} \setminus \{rn(a_1, \dots, a_n)\}$)

- **Union:** For two input dataset D_i^{in}, D_k^{in} with schema $rn(x_1, \dots, x_n)$, compute $D_j^{out} = D_i^{in} \cup D_k^{in}$
- **Join:** For two input dataset D_i^{in}, D_k^{in} with schema $rn_i(x_1, \dots, x_n)$ and $rn_k(y_1, \dots, y_q)$, for a chosen pair of indices ind_i, ind_k compute $D_j^{out} = \{rn'(x_1, \dots, x_n, y_1, \dots, y_{ind_k-1}, y_{ind_k+1}, \dots, y_q) \mid x_{ind_i} = y_{ind_k} \wedge rn_i(x_1, \dots, x_n) \in D_i^{in} \wedge rn_k(y_1, \dots, y_q) \in D_k^{in}\}$
- **Difference:** For two input dataset D_i^{in}, D_k^{in} with the same schema $rn(x_1, \dots, x_n)$, compute $D_j^{out} = D_i^{in} \setminus D_k^{in}$

WORKERS OPERATIONS: These are elementary tasks performed by workers to modify the datasets computed so far. These operations may perform non-deterministic choices among possible outputs.

- **Field addition:** Given an input dataset D_i^{in} with schema $rn(x_1, \dots, x_n)$, a predicate $P(\cdot)$, compute a dataset D_j^{out} with schema $rn'(x_1, \dots, x_n, x_{n+1})$ such that every tuple $rn(a_1, \dots, a_n) \in D_i^{in}$ is extended to a tuple $rn(a_1, \dots, a_n, a_{n+1}) \in D_j^{out}$ such that $P(a_1, \dots, a_{n+1})$ holds. Note that the value of field x_{n+1} can be chosen **non-deterministically** for each record.
- **Record input:** Given an input dataset D_i^{in} with schema $rn(x_1, \dots, x_n)$, and a predicate P , compute a dataset D_j^{out} on the same schema $rn(x_1, \dots, x_n)$ with an additional record $rn(a_1, \dots, a_n)$ such that $P(a_1, \dots, a_n)$ holds. Note that the value of fields $x_1, \dots, x_{n+1}$ can be non-deterministically chosen. Intuitively, P defines the set of possible entries in a dataset.
- **Field update:** For each record $rn(a_1 \dots a_n)$ of D_i^{in} , compute a record $rn(b_1, \dots, b_n)$ in D_j^{out} such that some linear arithmetic predicate $P(a_1, a_n, b_1, \dots, b_n)$ holds. Again, any value for $b_1, \dots, b_n$ that satisfies P can be chosen.

RECORD TO RECORD ARITHMETIC OPERATIONS: For each record $rn_i(a_1 \dots a_n)$ of D_i^{in} , compute a record in D_j^{out} of the form $rn_j(b_1, \dots, b_q)$ such that each $b_k, k \in 1..q$ is a linear combination of $a_1 \dots a_n$.

These operations can easily define tasks that split a database D_1^{in} in two datasets D_1^{out}, D_2^{out} , one containing records that satisfy some predicate P and the other one records that do not satisfy P . Similarly, one can describe input of record from a worker, operations that assemble datasets originating from different threads ... To summarize, when a node n with associated task t with I predecessors is executed, we can slightly abuse our notations and write $D_k^{out} = f_{k,t}(D_1^{in}, \dots, D_I^{in})$ when the task performs a deterministic calculus (SQL-based operations or record to record calculus), and $D_k^{out} \in F_{k,t}(D_1^{in}, \dots, D_I^{in})$ when the tasks involves non-deterministic choice of a worker. We now give the operational semantics of complex workflows.

An **execution** starts from the initial workflow W_0 (the initial high-level description provided by a requester). Executing a complex workflow consists in realizing all its tasks following the order given by the dependency relation $\longrightarrow$ in the orchestration, possibly after some refinement steps. At each step of an

execution, the remaining part of the workflow to execute, the assignments of tasks to workers and the data input to tasks are memorized in a *configuration*. Execution steps consist in updating configurations according to operational rules. They assign a task to a competent worker, execute an atomic or automated task (i.e. produce output data from input data), or refine a complex task. Executions end when the remaining workflow to execute contains only the final node n_f .

A *worker assignment* for a workflow $W = (N, \longrightarrow, \lambda)$ is a partial map $\mathbf{wa} : N \rightarrow \mathcal{U}$ that assigns a worker to a node in the workflow. Let $\mathbf{wa}(n) = w_i$. If $\lambda(n)$ is a complex task, then there exists a rule $r = (\lambda(n), W) \in \mathcal{R}$ such that $(w_i, r) \in sk$ (worker w_i knows how to refine task $\lambda(n)$). Similarly, if $\lambda(n)$ is an atomic task, then $(w_i, \lambda(n)) \in sk$ (worker w_i has the competences needed to realize $\lambda(n)$). We furthermore require map $\mathbf{wa}$ to be injective, i.e. a worker is involved in at most one task. We say that $w_i \in \mathcal{U}$ is free if $w_i \notin \mathbf{wa}(N)$. If $\mathbf{wa}(n)$ is not defined, and w_i is a free worker, $\mathbf{wa} \cup \{(n, w_i)\}$ is the map that assigns node n to worker w_i , and is unchanged for other nodes. Similarly, $\mathbf{wa} \setminus \{n\}$ is the restriction of $\mathbf{wa}$ to $N \setminus \{n\}$.

A *data assignment* for a workflow W is a function $\mathbf{Dass} : N \rightarrow (DB \uplus \{\emptyset\})^*$, that maps nodes in W to sequence of input datasets. For a node with k predecessors $n_1, \dots, n_k$, we have $\mathbf{Dass}(n) = D_1 \dots D_k$. A dataset D_i can be empty if n_i has not been executed yet, and hence has produced no data. $\mathbf{Dass}(n)_{[i/X]}$ is the sequence obtained by replacement of D_i by X in $\mathbf{Dass}(n)$.

Definition 6 (Configuration). A configuration of a complex workflow is a triple $C = (W, \mathbf{wa}, \mathbf{Dass})$ where W is a workflow depicting remaining tasks that have to be completed, $\mathbf{wa}$ is a worker assignment, and $\mathbf{Dass}$ is a data assignment.

A complex workflow execution starts from the *initial configuration* $C_0 = (W_0, \mathbf{wa}_0, \mathbf{Dass}_0)$, where $\mathbf{wa}_0$ is the empty map, $\mathbf{Dass}_0$ associates dataset D_{in} provided by client to n_{init} and sequences of empty datasets to all other nodes of W_0 . A *final configuration* is a configuration $C_f = (W_f, \mathbf{wa}_f, \mathbf{Dass}_f)$ such that W_f contains only node n_f , $\mathbf{wa}_f$ is the empty map, and $\mathbf{Dass}_f(n_f)$ represents the dataset that was assembled during the execution of all nodes preceding n_f and has to be returned to the client. The intuitive understanding of this type of configuration is that n_f needs not be executed, and simply terminates the workflow by returning final output data. Note that due to data assignment, there can be more than one final configuration, and we denote by $\mathcal{C}_f$ the set of all final configurations.

We define the operational semantics of a complex workflow with 4 rules that transform a configuration $C = (W, \mathbf{wa}, \mathbf{Dass})$ in a successor configuration $C' = (W', \mathbf{wa}', \mathbf{Dass}')$. Rule 1 defines task assignments to free workers, Rule 2 defines the execution of an atomic task by a worker, Rule 3 defines the execution of an automated task, and Rule 4 formalizes refinement. We only give an intuitive description of semantic rules and refer reader to [12] for a complete operational semantics.

Rule 1 (WORKER ASSIGNMENT): A worker $u \in \mathcal{U}$ is assigned to a node n . The rule applies if u is free, has the skills required by $t = \lambda(n)$, if t is not an

automated task ($t \notin \mathcal{T}_{aut}$) and if node n is not already assigned to a worker. Note that a worker can be assigned to a node even if it does not have input data yet, and is not yet executable. This rule only changes the worker assignment part in a configuration.

Rule 2 (ATOMIC TASK COMPLETION): An atomic task $t = \lambda(n)$ can be executed if node n is *minimal* in current workflow W , it is assigned to a worker $u = \mathbf{wa}(n)$ and its input data $\mathbf{Dass}(n)$ does not contain an empty dataset. Upon completion of task t , worker u publishes the produced data $\mathcal{D}^{out}$ to the succeeding nodes of n in the workflow and becomes available. This rule modifies the workflow part (node n is removed), the worker assignment, and the data assignment (new data is produced and made available to successors of n).

Rule 3 (AUTOMATED TASK COMPLETION): An automated task $t = \lambda(n)$ can be executed if node n is minimal in the workflow and its input data do not contain an empty dataset. The difference with atomic tasks completion is that n is not assigned a worker, and the produced outputs are a deterministic function of task inputs. Rule 3 modifies the workflow part (node n is removed), and the data assignment.

Rule 4 (COMPLEX TASK REFINEMENT): The refinement of a node n with $t = \lambda(n) \in \mathcal{T}_{cx}$ by worker $u = \mathbf{wa}(n)$ uses a refinement rule R_t such that $(u, t) \in sk$, and $R_t = (t, W_t)$. Rule 4 refines node n with workflow $W_t = (N_s, \longrightarrow_s, \lambda_s)$ (see Definition 4). Data originally used as input by n become inputs of the source node of W_t . All other newly inserted nodes have empty input datasets. This rule changes the workflow part of configurations and data assignment accordingly.

We say that there exists a *move* from a configuration C to a configuration C' , or equivalently that C' is a successor of configuration C and write $C \rightsquigarrow C'$ whenever there exists a rule that transforms C into C' .

Definition 7 (Run). A run $\rho = C_0.C_1 \dots C_k$ of a complex workflow is a finite sequence of configurations such that C_0 is an initial configuration, and for every $i \in 1 \dots k$, $C_{i-1} \rightsquigarrow C_i$. A run is maximal if C_k has no successor. A maximal run is terminated iff C_k is a final configuration, and it is deadlocked otherwise.

In the rest of the paper, we denote by $\mathcal{Runs}(CW, D_{in})$ the set of maximal runs originating from initial configuration $C_0 = (W_0, \mathbf{wa}_0, \mathbf{Dass}_0)$ (where $\mathbf{Dass}_0$ associates dataset D_{in} to node n_{init}). We denote by $\mathcal{Reach}(CW, D_{in})$ the set of configurations that can be reached from C_0 . Along a run, the size of datasets in use can grow, and the size of the workflow can also increase, due to refinement of tasks. Hence, $\mathcal{Reach}(CW, D_{in})$ and $\mathcal{Runs}(CW, D_{in})$ need not be finite. Indeed, complex tasks and their refinement can encode unbounded recursive schemes in which workflow parts or datasets grow up to arbitrary sizes. Even when $\mathcal{Reach}(CW, D_{in})$ is finite, a complex workflow may exhibit infinite cyclic behaviors. Hence, without restriction, complex workflows define transitions systems of arbitrary size, with growing data or workflow components, and with cycles.

In Sect. 5, we give an algorithm to check termination of a complex workflow with bounded recursion. Roughly speaking, this algorithm searches a reachable configuration C_{bad} in which emptiness of a dataset D could stop an execution. Once such a configuration is met, it remains to show that the statement $D = \emptyset$ is compatible with the operations performed by the run (insertion, projections, unions of datasets ...) before reaching C_{bad} . This is done by computing backward the weakest preconditions ensuring $D = \emptyset$ along the followed run, and checking their satisfiability. Weakest precondition were introduced in [17] to prove correctness of programs, and used recently to verify web applications with embedded SQL [20].

Definition 8. *Let $C \rightsquigarrow C'$ be a move. Let m be the nature of this move (an automated task realization, a worker assignment, a refinement ...). We denote by $wp[m]\psi$ the weakest precondition required on C such that ψ holds in C' after move m .*

Moves create dependencies among input and output datasets of a task, that are captured as FO properties. Further, the weakest precondition $wp[m]\psi$ of an FO property ψ is also an FO property.

Proposition 1. *Let CW be a complex workflow, r be the maximal arity of relational schemas in CW and ψ be an FO formula. Then for any move m of CW , $wp[m]\psi$ is an effectively computable FO formula, and is of size in $O(r \cdot |\psi|)$.*

A complete proof is available in [12]. We show in Sect. 5 that many properties of complex workflows need restrictions on recursion and forbid SQL difference to be decidable. Proposition 1 does not need such assumption. Weakest preconditions calculus is rather syntactic and is effective for any FO formula and any move. Now, if automated tasks use SQL difference, universal quantifiers can be introduced in existential blocks, leading to weakest preconditions in an FO fragment where satisfiability is undecidable. Interestingly, if automated tasks do not use SQL difference, a weakest precondition is mainly obtained by syntactic replacement of relational statements and changes of variables. It can increase the number of variables, but it does not change the number of quantifier blocks nor their ordering, and does not introduce new quantifiers when replacing an atom. We hence easily obtain:

Corollary 1. *The existential, universal, BSR and SF fragments of FO are closed under calculus of a weakest precondition if tasks do not use SQL difference.*

5 Termination of Complex Workflows

Complex workflows use the knowledge and skills of crowd workers to complete a task starting from input data provided by a client. However, a workflow may never reach a final configuration because data cannot be processed properly by the workflow, or because infinite recursive schemes appear during the execution.

Definition 9 (Deadlock, Termination). Let CW be a complex workflow, D_{in} be an initial dataset, $\mathcal{D}_{in}$ be a set of datasets. CW terminates existentially on input D_{in} iff there exists a run in $\text{Runs}(CW, D_{in})$ that is terminated. CW terminates universally on input D_{in} iff all runs in $\text{Runs}(CW, D_{in})$ are terminated. Similarly, CW terminates universally on input set $\mathcal{D}_{in}$ iff CW terminates universally on every input $D_{in} \in \mathcal{D}_{in}$, and CW terminates existentially on $\mathcal{D}_{in}$ iff some run of CW terminates for an input $D_{in} \in \mathcal{D}_{in}$.

When addressing termination for a set of inputs, we describe $\mathcal{D}_{in}$ symbolically with a decidable fragment of FO ($\forall\text{FO}$, $\exists\text{FO}$, BSR, or SF-FO). Complex workflows are Turing powerful (we show an encoding of a counter machine in [12]). So we get:

Theorem 1. *Existential termination of complex workflows is undecidable.*

An interesting point is that undecidability does not rely on arguments based on the precise contents of datasets (that are ignored by semantic rules). Indeed, execution of tasks only requires non-empty input datasets. Undecidability holds as soon as higher order operations (semantic rule 4) are used.

Universal termination is somehow an easier problem than existential termination. We show in this section that it is indeed decidable for many cases and in particular when the datasets used as inputs of a complex workflow are explicitly given or are specified in a decidable fragment of FO. We proceed in several steps. We first define symbolic configurations, i.e. descriptions of the workflow part of configurations decorated with relational schemas depicting data available as input of tasks. We define a successor relation for symbolic configurations. We then identify the class of non-recursive complex workflows, in which the length of executions is bounded by some value $K_{\mathcal{T}_{cx}}$. We show that for a given finite symbolic executions ρ^S , and a description of inputs, one can check whether there exists an execution ρ that coincides with ρ^S . This proof builds on the effectiveness of calculus of weakest preconditions along a particular run (see Proposition 1). Then, showing that a complex workflow does not terminate amounts to proving that it is not recursion-free, or that it has a finite symbolic run which preconditions allow a deadlock.

Definition 10 (Symbolic configuration). Let $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ be a complex workflow with database schema DB . A symbolic configuration of CW is a triple $C^S = (W, Ass, D_{ass}^S)$ where $W = (N, \rightarrow, \lambda)$ is a workflow, $Ass : N \rightarrow \mathcal{U}$ assigns workers to nodes, and $D_{ass}^S : N \rightarrow (DB)^*$ associates a list of relational schemas to nodes of the workflow.

Symbolic configuration describe the status of workflow execution as in standard configurations (see Definition 6) but leaves the data part underspecified. For every node n that is minimal in W , the meaning of $\mathbf{Dass}^S(n) = rs_1 \dots rs_k$ is that task attached to node n takes as inputs datasets $D_1 \dots D_k$ where each D_i conforms to relational schema rs_i . For a given symbolic configuration, we can find all rules that apply (there is only a finite number of worker assignments

or task executions), and compute successor symbolic configurations. This construction is detailed in [12].

Definition 11 (Deadlocks, Potential deadlocks). *A symbolic configuration $C^S = (W, Ass, Dass^S)$ is final if W consists of a single node n_f . It is a deadlock if it has no successor. It is a potential deadlock iff a task execution can occur from this node, i.e. there exists $n, \in \min(W)$ such that $\lambda(n)$ is an automated or atomic task.*

A deadlocked symbolic configuration represents a situation where progress is blocked due to shortage of competent workers to execute tasks. A potential deadlock is a symbolic configuration C^S where empty datasets may stop an execution. This deadlock is *potential* because $Dass^S$ does not indicate whether a particular dataset D_i is empty. We show in this section that one can decide whether a potential deadlock situation in C^S represents a real and reachable deadlock, by considering how the contents of dataset D_i is forged along the execution leading to C^S .

Definition 12 (Symbolic run). *A symbolic run is a sequence $\rho^S = C_0^S \xrightarrow{m_1} C_1^S \xrightarrow{m_2} \dots \xrightarrow{m_k} C_k^S$ where each C_i^S is a symbolic configuration, C_{i+1} is a successor of C_i , and $C_0^S = (W_0, Ass_0, \mathbf{Dass}^S)$ where W_0, Ass_0 have the same meaning as for configurations, and $Dass_0^S$ associates to the minimal node n_0 in W_0 the relational schema of $Dass_0(n_0)$.*

One can associate to every execution of a complex workflow $\rho = C_0 \xrightarrow{m_1} C_1 \dots C_k$ a symbolic execution $\rho^S = C_0^S \xrightarrow{m_1} C_1^S \xrightarrow{m_2} \dots \xrightarrow{m_k} C_k^S$ called its *signature* by replacing data assignment in each $C_i = (W_i, Ass_i, \mathbf{Dass}_i)$ by a function from each node n to the relational schemas of the datasets in $\mathbf{Dass}_i(n)$. It is not true, however, that every symbolic execution is the signature of an execution of CW , as some moves might not be allowed when a dataset is empty (this can occur for instance when datasets are split, or after a selection). A natural question when considering a symbolic execution ρ^S is whether it is the signature of an actual run of CW . The proposition below shows that the decidability of this question depends on assumptions on input datasets.

Proposition 2. *Let CW be a complex workflow, D_{in} be a dataset, $\mathcal{D}_{in}$ be an FO formula with n_{in} variables, and $\rho^S = C_0^S \dots C_k^S$ be a symbolic run. If tasks do not use SQL difference, then deciding if there exists a run ρ with input dataset D_{in} and signature ρ^S is in 2EXPTIME. Checking if there exists a run ρ of CW and an input dataset D_{in} that satisfies $\mathcal{D}_{in}$ with signature ρ^S is undecidable in general. If tasks do not use SQL difference, then it is in*

- 2EXPTIME if $\mathcal{D}_{in}$ is in $\exists FO$ and 3EXPTIME if $\mathcal{D}_{in}$ is in $\forall FO$ or BSR-FO.
- n_{in} -foldEXPTIME where n_{in} is the size of the formula describing $\mathcal{D}_{in}$ if $\mathcal{D}_{in}$ is in SF-FO

A proof of this Proposition is given in [12]. It does not yet give an algorithm to check termination, as a complex workflows may have an infinite set of runs

and runs of unbounded length. However, semantic rules 1, 2, 3 can be used a bounded number of times from a configuration (they decrease the number of available users or remaining tasks in W). Unboundedness then comes from the refinement rule 4, that implements recursive rewriting schemes.

Definition 13. Let t be a complex task. We denote by $\text{Rep}(t)$ the task names that can appear when refining task t , i.e. $\text{Rep}(t) = \{t' \mid \exists u, W, (u, t) \in \text{sk} \wedge (t, W) \in \mathcal{R} \wedge t' \in \lambda(N_W)\}$. The rewriting graph of a complex workflow $CW = (W_0, \mathcal{T}, \mathcal{U}, \text{sk}, \mathcal{R})$ is the graph $\text{RG}(CW) = (\mathcal{T}_{cx}, \longrightarrow_R)$ where $(t_1, t_2) \in \longrightarrow_R$ iff $t_2 \in \text{Rep}(t_1)$. Then CW is recursion-free if there is no cycle of $\text{RG}(CW)$ that is accessible from a task appearing in W_0 .

When a complex workflow is not recursion free, then some executions may exhibit infinite behaviors in which some task t_i is refined infinitely often. Such an infinite rewriting loop can contain a deadlock. In this case, the complex workflow does not terminate. If this infinite rewriting loop does not contain deadlocks, the complex workflow execution will never reach a final configuration in which all tasks have been executed. Hence, complex workflows that are not recursion-free do not terminate universally (see the proof of Proposition 3 for further explanations). Recursion-freeness is decidable and in $O(|\mathcal{T}_{cx}^2| + |\mathcal{R}|)$ (see [12] for details). Further, letting d denote the maximal number of complex tasks appearing in a rule, there exists a bound $K_{\mathcal{T}_{cx}} \leq d^{|\mathcal{T}_{cx}|}$ on the size of W in a configuration, and the length of a (symbolic) execution of CW is in $O(3 \cdot K_{\mathcal{T}_{cx}})$ (see [12]). We now characterize complex workflows that terminate universally.

Proposition 3. A complex workflow terminates universally if and only if:

- i) it is recursion free, ii) it has no (symbolic) deadlocked execution,
- iii) there exists no run with signature $C_0^S \dots C_i^S$ where C_i^S is a potential deadlock, with $D_k = \emptyset$ for some $D_k \in \mathbf{Dass}(n_j)$ and for some minimal node n_j of W_i .

Condition i) can be easily verified (see [12] for a proof). If i) holds, then there is a bound $3 \cdot K_{\mathcal{T}_{cx}}$ on the length of all symbolic executions, and checking condition ii) is an exploration of reachable symbolic configurations to find deadlocks. It requires $\log(3 \cdot K_{\mathcal{T}_{cx}}) \cdot K_{\mathcal{T}_{cx}}$ space, i.e. can be done in EXPTIME (w.r.t. $K_{\mathcal{T}_{cx}}$). Checking condition iii) is more involved. First, it requires finding a potential deadlock, i.e. a reachable symbolic execution C_i^S with a minimal node n_j representing a task to execute. Then one has to check if there exists an actual run $\rho = C_0 \dots C_i$ with signature $C_0^S \dots C_i^S$ such that one of the input datasets D_k in sequence $\mathbf{Dass}_i(n_j)$ is empty. Let $rs_j = (rn_k, A_k)$ be the relational schema of D_k , with $A_j = \{a_1, \dots, a_{|A_j|}\}$. Then, emptiness of D_k can be encoded as a universal FO formula of the form $\psi_k^* := \forall x_1, \dots, x_{|A_j|}, rn_k(x_1, \dots, x_{|A_j|}) \notin D_k$. Then all weakest preconditions needed to reach C_i with $D_k = \emptyset$ can be computed iteratively, and remain in the universal fragment of FO (see Proposition 1). If a precondition ψ_j computed this way (at step $j < i$) is unsatisfiable, then there is no actual run with signature ρ^S such that D_k is an empty dataset. If one ends weakest precondition calculus on configuration C_0^S with a condition ψ_0 that is

satisfiable, and $D_{in} \models \psi_0$, then such a run exists and *iii*) does not hold. Similarly with inputs described by $\mathcal{D}_{in}$, if $\mathcal{D}_{in} \wedge \psi_0$ is satisfiable, then condition *iii*) does not hold. Note that this supposes that $\mathcal{D}_{in}$ is written in a decidable FO fragment.

Theorem 2. *Let CW be a complex workflow, in which tasks do not use SQL difference. Let D_{in} be an input dataset, and $\mathcal{D}_{in}$ be an FO formula. Universal termination of CW on input D_{in} is in $co-2EXPTIME$. Universal termination on inputs that satisfy $\mathcal{D}_{in}$ is undecidable in general. It is in*

- $co-2EXPTIME$ (in K , the length of runs) if $\mathcal{D}_{in}$ is in $\forall FO$, $co-3EXPTIME$ if $\mathcal{D}_{in}$ is $\exists FO$ or $BSR-FO$
- $co-n_{in}\text{-fold-}EXPTIME$, where $n_{in} = |\mathcal{D}_{in}| + 2^K$ if $\mathcal{D}_{in}$ is in $SF-FO$.

One can notice that the algorithm to check universal termination of CWs guesses a path, and is hence non-deterministic. In the worst case, one may have to explore *all* symbolic executions of size at most $3 \cdot K_{\mathcal{T}_{cx}}$.

Undecidability of existential termination has several consequences: As complex workflows are Turing complete, automatic verification of properties such as reachability, coverability, boundedness of datasets, or more involved properties written in logics such as LTL FO [14] are also undecidable. However, the counter machine encoding in the proof of Theorem 1 uses recursive rewriting schemes. A natural question is then whether existential termination is decidable when all runs have a length bounded by some integer $K_{\max}$. Indeed, when such a bound exists it is sufficient to find a terminating witness, i.e. find a signature ρ^S of size $K_{\max}$ ending on a final configuration non-deterministically, compute weakest preconditions $\psi_{|\rho^S|}, \dots, \psi_0$ and check their satisfiability. The length of ψ_0 is in $O(r^{K_{\max}})$, and as one has to verify non-emptiness of datasets used as inputs of tasks to allow execution of a run with signature ρ^S , the preconditions are in the existential fragment of FO, yielding a satisfiability of preconditions in $O(2^{r^{K_{\max}}})$.

Theorem 3. *Let CW be complex workflow in which tasks do not use SQL difference, and which runs are of length $\leq K_{\max}$. Let D_{in} be a dataset, and $\mathcal{D}_{in}$ a FO formula. One can decide in $2-EXPTIME$ (in $K_{\max}$) whether CW terminates existentially on input D_{in} . If $\mathcal{D}_{in}$ is in $\exists FO$, termination of CW is also in $2-EXPTIME$. It is in $3-EXPTIME$ when $\mathcal{D}_{in}$ is in $\forall FO$ or $BSR-FO$*

Recursion-free CWs have runs of length bounded by $3 \cdot K_{\mathcal{T}_{cx}}$, so existential termination is decidable as soon as automated tasks do not use SQL difference (which makes alternations of quantifiers appear in weakest preconditions). The bound $K_{\mathcal{T}_{cx}}$ can be exponential (see proof of Proposition 2), but in practice, refinements are supposed to transform a complex task into an orchestration of simpler subtasks, and one can expect $K_{\mathcal{T}_{cx}}$ to be a simple polynomial in the number of complex tasks. Another way to bound recursion in a CW is to limit the number of refinements that can occur during an execution. For instance, if each complex task can be decomposed at most k times, the bound on length of runs becomes $K_{\max} = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$.

6 Correctness of Complex Workflows

Complex workflows provide a service to a client, that inputs some data (a dataset D_{in}) to a complex task, and expects some answer, returned as a dataset D_{out} . A positive answer to a termination question means that the process specified by a complex workflow does not deadlock in some/all executions. Yet, the returned data can still be incorrect. We assume the client sees the crowdsourcing platform as a black box, and simply asks for the realization of a complex task that needs specific competences. However, the client may have requirements on the type of output returned for a particular input. We express this constraint with a FO formula $\psi^{in,out}$ relating inputs and outputs, and extend the notions of existential and universal termination to capture the fact that a complex workflow implements client's needs if some/all runs terminate, and in addition fulfill requirements $\psi^{in,out}$. This is called *correctness*.

Definition 14. A constraint on inputs and outputs is an FO formula $\psi^{in,out} ::= \psi_E^{in,out} \wedge \psi_A^{in,out} \wedge \psi_{AE}^{in,out} \wedge \psi_{EA}^{in,out}$, where

- $\psi_E^{in,out}$ is a conjunction of $\exists$ FO formulas addressing the contents of the input/output dataset, of the form $\exists x, y, z, rn(x, y, z) \in D_{in} \wedge P(x, y, z)$, where $P(\cdot)$ is a predicate,
- $\psi_A^{in,out}$ is a conjunction of $\forall$ FO formulas constraining all tuples of the input/output dataset, of the form $\forall x, y, z, rn(x, y, z) \in D_{in} \Rightarrow P(x, y, z)$
- $\psi_{AE}^{in,out}$ is a conjunction of formulas relating the contents of inputs and outputs, of the form $\forall x, y, z, rn(x, y, z) \in D_{in} \Rightarrow \exists(u, v, t), \varphi(x, y, z, u, v, t)$, where φ is a predicate.
- $\psi_{EA}^{in,out}$ is a conjunction of formulas relating the contents of inputs and outputs, of the form $\exists x, y, z, rn(x, y, z) \in D_{in}, \forall(u, v, t), \varphi(x, y, z, u, v, t)$

The ψ_{AE} part of the I/O constraint can be used to require that every record in an input dataset is tagged in the output. The ψ_{EA} part can be used to specify that the output is a particular record selected from the input dataset (to require correctness of a workflow that implements a vote).

Definition 15 (Correctness). Let CW be a complex workflow, $\mathcal{D}_{in}$ be a set of input datasets, and $\psi^{in,out}$ be a constraint given by a client. A run in $\text{Runs}(CW, D_{in})$ is correct if it ends in a final configuration and returns a dataset D_{out} such that $D_{in}, D_{out} \models \psi^{in,out}$. CW is existentially correct with inputs $\mathcal{D}_{in}$ iff there exists a correct run $\text{Runs}(CW, D_{in})$ for some $D_{in} \in \mathcal{D}_{in}$. CW is universally correct with inputs $\mathcal{D}_{in}$ iff all runs in $\text{Runs}(CW, D_{in})$ are correct for every $D_{in} \in \mathcal{D}_{in}$.

In general, termination does not guarantee correctness. A terminated run starting from an input dataset D_{in} may return a dataset D_{out} such that pair D_{in}, D_{out} does not comply with constraint $\psi^{in,out}$ imposed by the client. For instance, a run may terminate with an empty dataset while the client required at least one answer. Similarly, a client may ask all records in the input dataset

to appear with an additional tag in the output. If any input record is missing, the output will be considered as incorrect. As for termination, correctness can be handled through symbolic manipulation of datasets, but has to consider constraints that go beyond emptiness of datasets. Weakest preconditions can be effectively computed (Proposition 1): one derives successive formulas $\psi_i^{in,out}, \dots, \psi_0^{in,out}$ between D_{in}, D_{out} and datasets in use at step $i, \dots, 0$ of a run. However, the $\psi_{AE}^{in,out}$ part of formulas is already in an undecidable fragment of FO, so even universal termination is undecidable in general, and even when a bound on the length of runs is known. It becomes decidable only with some restrictions on the fragment of FO used to write $\psi^{in,out}$.

Theorem 4. *Existential and universal correctness of CW are undecidable, even when runs are of bounded length K . If tasks do not use SQL difference, and $\psi^{in,out}$ is in a decidable fragment of FO, then*

- *existential correctness is decidable for CWs with runs of bounded length, and is respectively in $2EXPTIME$, $3EXPTIME$ and 2^K -fold- $EXPTIME$ for the $\exists FO$, $\forall FO$, BSR, and SF fragments.*
- *universal correctness is decidable and is respectively in co- $2EXPTIME$, co- $3EXPTIME$ and co- 2^K -fold- $EXPTIME$ for the $\forall FO$, $\exists FO$, BSR, and SF fragments.*

Proof. (Sketch). First, a CW that does not terminate (existentially or universally) cannot be correct, and setting $\psi^{in,out}::=true$ we get a termination question. So existential correctness is undecidable for any class of input/output constraint. Then, if $\psi^{in,out}$ is in a decidable fragment of FO, and operations do not use SQL difference, then weakest preconditions preserve this fragment, and we can apply the algorithm used for Theorem 2 and 3, starting from precondition $\psi|_{\rho^S} = \psi^{in,out}$. $\square$

Restricting constraints to $\exists FO$, $\forall FO$, BSR, or SF-FO can be seen as a limitation. However, $\exists FO$ can already express non-emptiness properties: $\exists x_1, \dots, \exists x_k, rn(x_1, \dots, x_k) \in D_{out}$ says that the output should contain at least one record. Now, to impose that every input is processed correctly, one needs a formula of the form $\psi_{in,out}^{valid}::= \forall x_1, \dots, x_k, rn(x_1, \dots, x_k) \in D_{in} \implies \exists y_1, \dots, y_q, rn(x_1, \dots, x_k, y_1, \dots, y_q) \in D_{out} \wedge P(x_1, \dots, x_k, y_1, \dots, y_q)$, that asks that every input in D_{in} appears in the output, and $P()$ describes correct outputs. Clearly, $\psi_{in,out}^{valid}$ is not in the separated fragment of FO. We can decide correctness for formulas $\psi_{AE}^{in,out}$ of the form $\forall \overrightarrow{X_1} \exists \overrightarrow{Y_2}, \varphi$ as soon as every atom in φ that is not separated contains only existential variables that take values from a finite domain. Then $\psi_{AE}^{in,out}$ can be transformed into an $\forall FO$ formula which matrix is a boolean combination of separated atoms.

7 Conclusion

We have proposed *complex workflows*, a model for crowdsourcing applications which enables intricate data centric processes built on higher order schemes and studied their termination and correctness. Unsurprisingly, termination of a complex workflow is undecidable. Now the question of whether *all* runs terminate can be answered when the initial data is specified in a fragment of FO for which satisfiability is decidable. Similar remarks apply to correctness. Table 1 below summarizes the complexities of termination and correctness for static complex workflows (without higher order answer) or with bounded recursion, and for generic workflows with higher order. We consider complexity of termination and correctness for different decidable FO fragments. The (co)-2EXPTIME bound for the fragments with the lowest complexity mainly comes from the exponential size of the formula depicting preconditions that must hold at initial configuration (the EXPTIME complexity is in the maximal length of runs). This can be seen as an untractable complexity, but one can expect depth of recursion to be quite low, or even enforce such a depth.

Table 1. Complexity of termination and correctness (*EXPT* stands for EXPTIME).

Workflow Type	FO Fragment (for $\mathcal{D}_{in}$ or $\psi^{in,out}$)	Problems & Complexity (no SQL diff.)			
		Existential Termination	Universal Termination	Existential Correctness	Universal Correctness
Static, Recursive Bounded	FO	Undecidable	Undecidable	Undecidable	Undecidable
	$\exists^*(\forall^*$ if univ. PB)	$2EXPT$	$co - 2EXPT$	$2EXPT$	$co - 2EXPT$
	$BSR, \forall^*(\exists^*$ if univ. PB)	$3EXPT$	$co - 3EXPT$	$3EXPT$	$co - 3EXPT$
	SF	$n_{in} - foldEXPT$	$co - n_{in} - foldEXPT$	$2^{K\tau_{cs}} - foldEXPT$	$co - 2^{K\tau_{cs}} - foldEXPT$
Recursive Unbounded	FO	Undecidable	Undecidable	Undecidable	Undecidable
	$\exists^*(\forall^*$ if univ. PB)	Undecidable	$co - 2EXPT$	Undecidable	$co - 2EXPT$
	$BSR, \forall^*(\exists^*$ if univ. PB)	Undecidable	$co - 3EXPT(K)$	Undecidable	$co - 3EXPT$
	SF	Undecidable	$co - n_{in} - foldEXPT$	Undecidable	$co - 2^{K\tau_{cs}} - foldEXPT$

Several questions remain open: So far, we do not know whether the complexity bounds are sharp. Beyond complexity issues, crowdsourcing relies heavily on incentives to make sure that a task progresses. Placing appropriate incentives to optimize the overall cost of a complex workflow and ensure progress in an important topic. Similarly, a crux in crowdsourcing is monitoring, in particular to propose tasks to the most competent workers. Cost optimization and monitoring can be addressed as a kind of quantitative game. Other research directions deal with the representation and management of imprecision. So far, there is no measure of trust nor plausibility on values input by workers during a complex workflow execution. Equipping domains with such measures is a way to provide control techniques targeting improvement of trust in answers returned by a complex workflow, and trade-offs between performance and accuracy of answers.

References

1. Amazon’s Mechanical Turk. <http://www.mturk.com>
2. Crowdfunder. <http://www.crowdfunder.com>
3. Cruk’s trailblazer. <http://www.cancerresearchuk.org>
4. Foule factory. <http://www.foulefactory.com>
5. Galaxy zoo. <http://zoo1.galaxyzoo.org>
6. Spipoll. <http://www.spipoll.org>
7. Abiteboul, S., Segoufin, L., Vianu, V.: Static analysis of active XML systems. *Trans. Database Syst.* **34**(4), 23:1–23:44 (2009)
8. Abiteboul, S., Vianu, V.: Collaborative data-driven workflows: think global, act local. In: *Proceedings of PODS 2013*, pp. 91–102. ACM (2013)
9. Badouel, E., Hlout, L., Morvan, C.: Petri nets with structured data. *Fundam. Inform.* **146**(1), 35–82 (2016)
10. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: *Proceedings of VLDB 2006*, pp. 343–354. ACM (2006)
11. Bernays, P., Schnfinkel, M.: Zum entscheidungsproblem der mathematischen logik. *Math. Ann.* **99**(1), 342–372 (1928)
12. Bourhis, O., Hlout, L., Miklos, Z., Singh, R.: Data centric workflows for crowd-sourcing. Technical report, Univ. Rennes, INRIA, (2019). <https://hal.inria.fr/hal-02508838>
13. Codd, E.F.: Relational completeness of data base sublanguages. *Database Systems*, pp. 65–98 (1972)
14. Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. *Trans. Database Syst.* **37**(3), 22 (2012)
15. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: Trujillo, J.C., et al. (eds.) *ER 2018*. LNCS, vol. 11157, pp. 219–235. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00847-5_17
16. Deutsch, A., Sui, L., Vianu, V., Zhou, D.: Verification of communicating data-driven web services. In: *Proceedings of PODS 2006*, pp. 90–99. ACM (2006)
17. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of program. *Commun. ACM* **18**(8), 453–457 (1975)
18. Garcia-Molina, H., Joglekar, M., Marcus, A., Parameswaran, A., Verroios, V.: Challenges in data crowdsourcing. *Trans. Knowl. Data Eng.* **28**(4), 901–911 (2016)
19. Hariri, B.B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: *Proceedings of PODS 2013*, pp. 163–174 (2013)
20. Itzhaky, S., et al.: On the automated verification of web applications with embedded SQL. In: *Proceedings of ICDT 2017, LIPIcs*, vol. 68, pp. 16:1–16:18 (2017)
21. Jensen, K.: Coloured Petri nets: a high level language for system design and analysis. In: Rozenberg, G. (ed.) *ICATPN 1989*. LNCS, vol. 483, pp. 342–416. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_31
22. Khashiyani, L.G.: Polynomial algorithms in linear programming. *U.S.S.R. Comput. Math. Math. Phys.* **20**, 51–68 (1980)
23. Kitchin, D., Cook, W.R., Misra, J.: A language for task orchestration and its semantic properties. In: Baier, C., Hermanns, H. (eds.) *CONCUR 2006*. LNCS, vol. 4137, pp. 477–491. Springer, Heidelberg (2006). https://doi.org/10.1007/11817949_32

24. Kittur, A., Smus, B., Khamkar, S., Kraut, R.E.: Crowdforge: crowdsourcing complex work. In: Proceedings of UIST 2011, pp. 43–52. ACM (2011)
25. Koutsos, A., Vianu, V.: Process-centric views of data-driven business artifacts. *J. Comput. Syst. Sci.* **86**, 82–107 (2017)
26. Kucherbaev, P., Daniel, F., Tranquillini, S., Marchese, M.: Crowdsourcing processes: a survey of approaches and opportunities. *IEEE Internet Comput.* **20**(2), 50–56 (2016)
27. Kulkarni, A., Can, M., Hartmann, B.: Collaboratively crowdsourcing workflows with turkomatic. In: Proceedings of CSCW 2012, pp. 1003–1012. ACM (2012)
28. Lazic, R., Newcomb, T., Ouaknine, J., Roscoe, A.W., Worrell, J.: Nets with tokens which carry data. *Fundam. Inform.* **88**(3), 251–274 (2008)
29. Lewis, H.R.: Complexity results for classes of quantificational formulas. *J. Comput. Syst. Sci.* **21**(3), 317–353 (1980)
30. Li, G., Wang, J., Zheng, Y., Franklin, M.J.: Crowdsourced data management: a survey. *Trans. Knowl. Data Eng.* **28**(9), 2296–2319 (2016)
31. Little, G., Chilton, L.B., Goldman, M., Miller, R.C.: Turkkit: tools for iterative tasks on mechanical turk. In: Proceedings of HCOMP 2009, pp. 29–30. ACM (2009)
32. Lomazova, I.A., Schnoebelen, P.: Some decidability results for nested Petri nets. In: Bjørner, D., Broy, M., Zamulin, A.V. (eds.) PSI 1999. LNCS, vol. 1755, pp. 208–220. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46562-6_18
33. Löwenheim, L.: Über möglichkeiten im relativkalkül. *Math. Ann.* **76**(4), 447–470 (1915)
34. Mavridis, P., Gross-Amblard, D., Miklós, Z.: Using hierarchical skills for optimized task assignment in knowledge-intensive crowdsourcing. In: Proceedings of WWW 2016, pp. 843–853. ACM (2016)
35. Mortimer, M.: On languages with two variables. *Math. Log. Q.* **21**(1), 135–140 (1975)
36. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. *IBM Syst. J.* **42**(3), 428–445 (2003)
37. OASIS. Web Services Business Process Execution Language. Technical report, OASIS (2007). <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>
38. Sturm, T., Voigt, M., Weidenbach, C.: Deciding first-order satisfiability when universal and existential variables are separated. In: Proceedings of LICS 2016, pp. 86–95 (2016)
39. Tranquillini, S., Daniel, F., Kucherbaev, P., Casati, F.: Modeling, enacting, and integrating custom crowdsourcing processes. *TWEB* **9**(2), 7:1–7:43 (2015)
40. van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., La Rosa, M., Mendling, J.: Correctness-Preserving configuration of business process models. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 46–61. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78743-3_4
41. Van Der Aalst, W.M.P., et al.: Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp. Comput.* **23**(3), 333–363 (2011)
42. Zheng, Q., Wang, W., Yu, Y., Pan, M., Shi, X.: Crowdsourcing complex task automatically by workflow technology. In: Cao, J., Liu, J. (eds.) MIPaC 2016. CCIS, vol. 686, pp. 17–30. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-3996-6_2