Data Centric Workflows for Crowdsourcing

Bourhis, Pierre[1] and Hélouët, Loïc[2] and Miklos, Zoltan[3] and Singh, Rituraj[3]

[1]CNRS, Univ. Lille and [2]INRIA Rennes and [3]Univ. Rennes, France

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 [25]. 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 [38], 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 [17, 29]. Crowdforge uses Map-Reduce techniques to solve complex tasks [23]. Turkit [30] builds on an imperative language embedding calls to services of a crowdsourcing platform, which requires programming skills to design complex orchestrations. Turkomatic [26] and [41] 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, 39], a variant of Petri nets, address the control part of business processes, but data is not central. Orchestration models such as ORC [22] or BPEL [36] 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 [20] 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 [14]), but they cannot model evolution of unbounded datasets. Other variants of PNs with tokens that carry data have also been proposed in [27]. 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 [31] 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 [35], and verification mechanisms for LTL-FO (LTL formula embedding FO statements over universally quantified variables) were proposed in [13, 24]. LTL-FO is decidable for subclasses of artifacts with data dependencies and arithmetic. [15] 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. [18] 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.

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 section 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

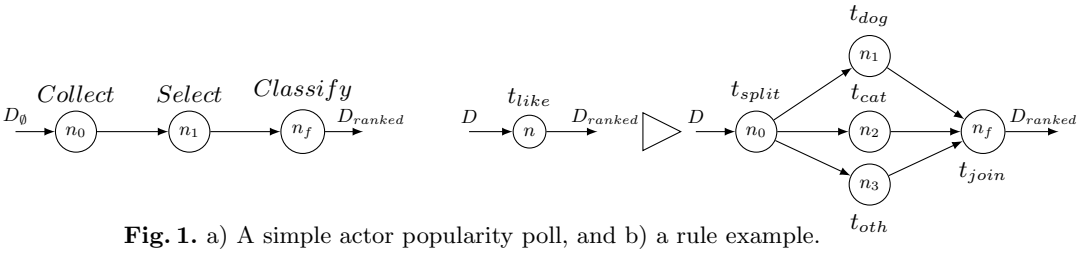

Fig. 1. a) A simple actor popularity poll, and b) a rule example.

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 Figure 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 (Sec. 4). We then address termination (Sec. 5) and correctness (Sec. 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 [12], i.e., data is organized in *datasets*, that follow *relational schemas*. We assume finite set of domains $\mathbf{dom} = dom_1, \ldots, dom_s$, a finite set of attribute names **att** and a finite set of relation names **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 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* $\overrightarrow{X} = x_1, \dots, x_n$ *is a formula of the form* $\varphi ::= \alpha(\overrightarrow{X}).\psi(\overrightarrow{X})$ *where* $\alpha(\overrightarrow{X})$ *is an alternation of quantifiers and variables of* $\overrightarrow{X}$*, i.e. sentences of the form* $\forall x_1 \exists x_2, \dots$ *called the* prefix *of* φ *and* $\psi(\overrightarrow{X})$ *is a quantifier free formula called the* matrix *of* φ*.* $\psi(\overrightarrow{X})$ *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 [21] if variables values are reals. Let $\overrightarrow{X_1} = \{x_1, \dots x_k\} \subseteq \overrightarrow{X}$. We write $\forall \overrightarrow{X_1}$ instead of $\forall x_1 . \forall x_2 \dots \forall x_k$, and $\exists \overrightarrow{X_1}$ instead of $\exists x_1 . \exists x_2 \dots \exists x_k$. We use w.l.o.g. formulas of the form $\forall \overrightarrow{X_1} \exists \overrightarrow{X_2} \dots \psi(X)$ or $\exists \overrightarrow{X_1} \forall \overrightarrow{X_2} \dots \psi(X)$, where $\psi(X)$ is a quantifier free matrix, and for every $i \neq j$, $\overrightarrow{X_i} \cap \overrightarrow{X_j} = \emptyset$. Every set of variables $\overrightarrow{X_i}$ 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 $\overrightarrow{X}$ 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 $x \in \overrightarrow{X}$.

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 $\forall \overrightarrow{X}.\varphi$ (resp. $\exists \overrightarrow{Y}.\varphi$) where φ is

quantifier free. We denote by ∀FO the universal fragment of FO and by ∃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 Appendix B 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 [34]. 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) [28].

Recent results [37] 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 $\forall\vec{X}_1.\exists\vec{Y}_2.\ldots\forall\vec{X}_n.\exists\vec{Y}_n\phi$, where $\vec{X}_1 \cdots \cup \vec{X}_n$ and $\vec{Y}_1 \cdots \cup \vec{Y}_n$ are separated. The SF fragment is powerful and subsumes the *Monadic Fragment* [32] (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 [37], 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 [19]. We show in Section 4 that conditions needed for a non-terminating execution of a complex workflow are in $\forall FO$, and that ∀FO, ∃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 [13], 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 $(n_1, n_2) \in \longrightarrow$, 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 $succ(n_i)$ the set of successors of a node n_i, and by $pred(n_i)$ 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 $W \backslash \{n_i\}$ the restriction of W to $N \backslash \{n_i\}$, 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* $W = (N, \longrightarrow, \lambda)$ *be a workflow,* $W' = (N', \longrightarrow', \lambda')$ *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* $W_{[n/W']} = (N_{[n/W']}, \longrightarrow_{[n/W']}, \lambda_{[n/W']})$*, where* $N_{[n/W']} = (N \setminus \{n\}) \cup N'$*,* $\lambda_{[n/W']}(n) = \lambda(n)$ *if* $n \in N, \lambda'(n)$ *otherwise, and*

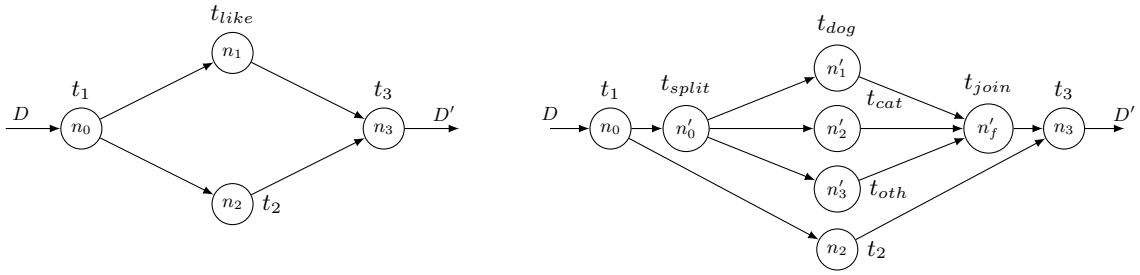

Fig. 2. A refinement of node n_1 using rule (t_{like}, W_{like}) of Figure 1-b.

$\rightarrow_{[n/W']} = \rightarrow' \cup \{(n_1, n_2) \in \rightarrow \mid n_1 \neq n \wedge n_2 \neq n\} \cup \{(n_1, n'_{src}) \mid (n_1, n) \in \rightarrow\} \cup \{(n'_f, n_2) \mid (n, n_2) \in \rightarrow\}$

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, 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 $src(W)$. 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 [33] for further studies on competences and more elaborated competence models.

Let us consider the example of Figure 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 Figure 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 Figure 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 $rs_1^{in}, \ldots rs_k^{in}$ (resp. $rs_1^{out}, \ldots rs_k^{out}$). We denote by $\mathcal{D}^{in} = D_1^{in}, \ldots D_k^{in}$ the set of datasets provided by predecessors of t, and by $\mathcal{D}^{out} = D_1^{out} \ldots D_q^{out}$ 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 $rn(x_1, \ldots, x_n)$ and a predicate $P(x_1, \ldots x_n)$, compute $D_j^{out} = \{rn(x_1, \ldots x_n) \mid rn(x_1, \ldots x_n) \in D_i^{in} \wedge P(x_1, \ldots x_n)\}$.
- **Projection:** For a given input dataset D_i^{in} with schema $rn(x_1, \ldots x_n)$ and an index $k \in 1..n$ compute $D_j^{out} = \{rn(x_1, \ldots x_{k-1}, x_{k+1}, \ldots x_n) \mid rn(x_1, \ldots x_n) \in D_i^{in}\}$.
- **Insertion/deletion** : For a given input dataset D_i^{in} and a fixed tuple $rn(a_1, \ldots a_n)$, compute $D_j^{out} = D_i^{in} \cup \{rn(a_1, \ldots a_n)\}$ (resp. $D_j^{out} = D_i^{in} \setminus \{rn(a_1, \ldots a_n)\}$)
- **Union:** For two input dataset D_i^{in}, D_k^{in} with schema $rn(x_1, \ldots x_n)$, compute $D_j^{out} = D_i^{in} \cup D_k^{in}$
- **Join:** For two input dataset D_i^{in}, D_k^{in} with schema $rn_i(x_1, \ldots x_n)$ and $rn_k(y_1, \ldots y_q)$, for a chosen pair of indices ind_i, ind_k compute $D_j^{out} = \{rn'(x_1, \ldots x_n, y_1, \ldots y_{ind_k-1}, y_{ind_k+1}, \ldots y_q) \mid x_{ind_i} = y_{ind_k} \wedge rn_i(x_1, \ldots x_n) \in D_i^{in} \wedge rn_k(y_1, \ldots y_q) \in D_k^{in}\}$
- **Difference:** For two input dataset D_i^{in}, D_k^{in} with the same schema $rn(x_1, \ldots x_n)$, 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 $rn(x_1, \ldots x_n)$, a predicate $P(.)$, compute a dataset D_j^{out} with schema $rn'(x_1, \ldots x_n, x_{n+1})$ such that every tuple $rn(a_1, \ldots, a_n) \in D_i^{in}$ is extended to a tuple $rn(a_1, \ldots, a_n, a_{n+1}) \in D_j^{out}$ such that $P(a_1, \ldots a_{n+1})$ 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 $rn(x_1, \ldots x_n)$, and a predicate P, compute a dataset D_j^{out} on the same schema $rn(x_1, \ldots x_n)$ with an additional record $rn(a_1, \ldots, a_n)$ such that $P(a_1, \ldots, a_n)$ holds. Note that the value of fields $x_1, \ldots x_{n+1}$ can be non-deterministically chosen. Intuitively, P defines the set of possible entries in a dataset.
- **Field update:** For each record $rn(a_1 \ldots a_n)$ of D_i^{in}, compute a record $rn(b_1, \ldots b_n)$ in D_j^{out} such that some linear arithmetic predicate $P(a_1, a_n, b_1, \ldots b_n)$ holds. Again, any value for $b_1, \ldots b_n$ that satisfies P can be chosen.

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

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 $D_k^{out} = f_{k,t}(D_1^{in}, \ldots D_I^{in})$

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 $\mathbf{wa}: N \to \mathcal{U}$ that assigns a worker to a subset of nodes 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, $\mathbf{wa} \backslash \{n\}$ is the restriction of $\mathbf{wa}$ to $N \setminus \{n\}$.

A *data assignment* for a workflow W is a function $\mathbf{Dass} : N \to (DB \uplus \{\emptyset\})^*$, 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 appendix A for a formal definition.

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 does 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. This rule 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 Def. 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 $\mathcal{R}uns(CW, D_{in})$ 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 $\mathcal{R}each(CW, D_{in})$ 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, $\mathcal{R}each(CW, D_{in})$ and $\mathcal{R}uns(CW, D_{in})$ 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 $\mathcal{R}each(CW, D_{in})$ 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 section 5, we give an algorithm to check termination of a complex workflow with bounded recursion. Roughly speaking, this algorithms 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 that they are satisfiable. Weakest precondition were introduced in [16] as a way to prove correctness of programs. They were used recently to verify web applications with embedded SQL [19].

Definition 8. *Let $C \rightsquigarrow C'$ be a move of a complex workflow. 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 for 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 $O(r.|\psi|)$.*

We refer readers to Appendix B.1 for a proof of Proposition 1. If automated tasks use SQL difference, universal quantifiers can be introduced in the weakest precondition. Interestingly, if automated tasks do not use SQL difference, a weakest precondition is mainly obtained by syntactic replacement of relational statements and by 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. This can be due to particular data input by workers that cannot be processed properly by the workflow, or to infinite recursive schemes appearing 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 $\mathcal{R}uns(CW, D_{in})$ that is terminated. CW* terminates universally *on input D_{in} iff all runs in $\mathcal{R}uns(CW, D_{in})$ 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 $\mathcal{D}_{in}$, we describe $\mathcal{D}_{in}$ symbolically with a decidable fragment of FO ($\forall$FO, $\exists$FO, BSR, or SF-FO). Complex workflows are Turing powerful (we show in Appendix C.1 how to encode a counter machine). 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 Prop. 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 $C^S = (W, Ass, Dass^S)$ where $W = (N, \rightarrow, \lambda)$ is a workflow, $Ass : N \rightarrow \mathcal{U}$ assigns workers to nodes, and $Dass^S : N \rightarrow (DB)^*$ associates a list of relational schemas to nodes of the workflow.*

Symbolic configuration describe the status of workflow execution as in standard configurations (see def. 6) but leaves the data part underspecified. For every node n that is minimal in W, the meaning of $\mathbf{Dass}^S(n) = rs_1, \ldots rs_k$ is that task attached to node n takes as inputs datasets $D_1 \ldots D_k$ 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 Appendix C.2.

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* $\rho^S = C_0^S \dots C_i^S$ *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*

This proposition is proved in Appendix C.3. 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* $Rep(t)$ *the task names that can appear when refining task* t*, i.e.* $Rep(t) = \{t' \mid \exists u, W, (u,t) \in sk \wedge (t, W) \in \mathcal{R} \wedge t' \in \lambda(N_W)\}$*. The* rewriting graph *of a complex workflow* $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ *is the graph* $RG(CW) = (\mathcal{T}_{cx}, \longrightarrow_R)$ *where* $(t_1, t_2) \in \longrightarrow_R$ *iff* $t_2 \in Rep(t_1)$*. Then* CW *is* recursion-free *if there is no cycle of* $RG(CW)$ *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 Prop. 3 for further explanations). We show in Appendix C.4, Prop. 5 that recursion-freeness is decidable and in $O(|\mathcal{T}_{cx}^2|+|\mathcal{R}|)$. 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 $O(3.K_{\mathcal{T}_{cx}})$ (see Prop. 6 in Appendix C.4). 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, as shown in Prop. 5, Appendix C.4. If *i)* holds, then there is a bound $3.K_{\mathcal{T}_{cx}}$ on the length of all symbolic executions, and checking condition *ii)* is an exploration of reachable symbolic configurations to find deadlocks. It requires $\log(3.K_{\mathcal{T}_{cx}}).K_{\mathcal{T}_{cx}}$ 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 $\psi_i ::= \forall x_1, \dots x_{|A_j|}, rn_k(x_1, \dots x_{|A_j|}) \notin D_k$. 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 Prop. 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*
- *$co-n_{in}$-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 [13] 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 $\psi_{|\rho^S|}, \ldots, \psi_0$ 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 $3.K_{\mathcal{T}_{cx}}$, 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 Prop. 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 from* $\exists x, y, z, rn(x, y, z) \in D_{in} \wedge P(x, y, z)$*, where* $P(.)$ *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* $\mathcal{R}uns(CW, D_{in})$ *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* $\mathcal{R}uns(CW, D_{in})$ *for some* $D_{in} \in \mathcal{D}_{in}$*.* CW *is universally correct with inputs* $\mathcal{D}_{in}$ *iff all runs in* $\mathcal{R}uns(CW, D_{in})$ *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 (Prop. 1): one derives successive formulas $\psi_i^{in,out}, \ldots \psi_0^{in,out}$ between D_{in}, D_{out} and datasets in use at step $i, \ldots 0$ 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 Thm. 2 and 3, starting from precondition $\psi_{|\rho^S|} = \psi^{in,out}$.□

Restricting constraints to ∃FO, ∀FO, BSR, or SF-FO can be seen as a limitation. However, ∃FO can already express non-emptiness properties: $\exists x_1, \ldots, \exists x_k$, $rn(x_1, \ldots, x_k) \in D_{out}$ 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 $\psi^{valid}_{in,out} ::= \forall x_1, \ldots, x_k, rn(x_1, \ldots x_k) \in D_{in} \Longrightarrow \exists y_1, \ldots, y_q, rn(x_1, \ldots x_k, y_1, \ldots y_q) \in D_{out} \wedge P(x_1, \ldots x_k, y_1, \ldots y_q)$, that asks that every input in D_{in} appears in the output, and $P()$ describes correct outputs. Clearly, $\psi^{valid}_{in,out}$ is not in the separated fragment of FO. We can decide correctness for formulas $\psi^{in,out}_{AE}$ 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 $\psi^{in,out}_{AE}$ can be transformed into an ∀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. We studied termination and correctness of complex workflows w.r.t. requirement on inputs and output of the overall process. Unsurprisingly, termination of a complex workflow is undecidable, already due to the control part of the model. 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 see as an untractable complexity, but one can however expect depth of recursion to be quite low, or even enforce such a depth.

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.

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)	$2EXPT$	$co-2EXPT$	$2EXPT$	$co-2EXP$
	BSR,$\forall^*(\exists^*$ if univ. PB)	$3EXPT$	$co-3EXPT$	$3EXPT$	$co-3EXPT$
	SF	$n_{in}-foldEXPT$	$co-n_{in}$-fold-$EXPT$	$2^{K\tau_{cx}}$-fold-$EXPT$	co-$2^{K\tau_{cx}}$-fold-$EXPT$
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-$2^{K\tau_{cx}}$-fold-$EXPT$

Table 1. Complexity of Termination and Correctness ($EXPT$ stands for EXPTIME).

References

1. Amazon's Mechanical Turk. http://www.mturk.com.
2. Crowdflower. http://www.crowdflower.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. S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active XML systems. *Trans. on Database Systems*, 34(4):23:1–23:44, 2009.
8. S. Abiteboul and V. Vianu. Collaborative data-driven workflows: think global, act local. In *Proc. of PODS'13*, pages 91–102. ACM, 2013.
9. E. Badouel, L. Hélouët, and C. Morvan. Petri nets with structured data. *Fundamenta Informaticae*, 146(1):35–82, 2016.
10. C. Beeri, A. Eyal, S. Kamenkovich, and Tova Milo. Querying business processes. In *Proc. of VLDB'06*, pages 343–354. ACM, 2006.
11. P. Bernays and M. Schönfinkel. Zum entscheidungsproblem der mathematischen logik. *Mathematische Annalen*, 99(1):342–372, 1928.
12. E. F. Codd. Relational completeness of data base sublanguages. *Database Systems*, pages 65–98, 1972.
13. E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *Trans. on Database Systems*, 37(3):22, 2012.
14. M. de Leoni, P. Felli, and M. Montali. A holistic approach for soundness verification of decision-aware process models. In *Proc. of ER'18*, volume 11157 of *LNCS*, pages 219–235, 2018.
15. A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proc. of PODS'06*, pages 90–99. ACM, 2006.
16. E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of program. *Communications of the ACM*, 18(8):453–457, 1975.
17. H. Garcia-Molina, M. Joglekar, A. Marcus, A. Parameswaran, and V. Verroios. Challenges in data crowdsourcing. *Trans. on Knowledge and Data Engineering*, 28(4):901–911, 2016.
18. B.B Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In *Proc. of PODS 2013*, pages 163–174, 2013.

19. S. Itzhaky, T. Kotek, N. Rinetzky, M. Sagiv, O. Tamir, H. Veith, and F. Zuleger. On the automated verification of web applications with embedded SQL. In *Proc. of ICDT'17*, volume 68 of *LIPIcs*, pages 16:1–16:18, 2017.
20. K. Jensen. Coloured petri nets: A high level language for system design and analysis. In *Proc. of Petri Nets'90*, volume 483 of *LNCS*, pages 342–416, 1989.
21. L.G. Khashiyan. Polynomial algorithms in linear programming. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 20:51–68, 1980.
22. D. Kitchin, W.R. Cook, and J. Misra. A language for task orchestration and its semantic properties. In *Proc. of CONCUR'06*, pages 477–491, 2006.
23. A. Kittur, B. Smus, S. Khamkar, and R.E. Kraut. Crowdforge: Crowdsourcing complex work. In *Proc. of UIST'11*, pages 43–52. ACM, 2011.
24. A. Koutsos and V. Vianu. Process-centric views of data-driven business artifacts. *Journal of Computer and System Sciences*, 86:82–107, 2017.
25. P. Kucherbaev, F. Daniel, S. Tranquillini, and M. Marchese. Crowdsourcing processes: A survey of approaches and opportunities. *IEEE Internet Computing*, 20(2):50–56, 2016.
26. A. Kulkarni, M. Can, and B. Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *Proc. of CSCW'12*, pages 1003–1012. ACM, 2012.
27. R. Lazic, T. Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fundamenta Informaticae*, 88(3):251–274, 2008.
28. H.R. Lewis. Complexity results for classes of quantificational formulas. *J. Comput. Syst. Sci.*, 21(3):317–353, 1980.
29. G. Li, J. Wang, Y. Zheng, and M.J. Franklin. Crowdsourced data management: A survey. *Trans. on Knowledge and Data Engineering*, 28(9):2296–2319, 2016.
30. G. Little, L.B. Chilton, M. Goldman, and R.C. Miller. Turkit: tools for iterative tasks on Mechanical Turk. In *Proc. of HCOMP'09*, pages 29–30. ACM, 2009.
31. I.A. Lomazova and Ph. Schnoebelen. Some decidability results for nested Petri nets. In *Perspectives of System Informatics*, pages 208–220, 1999.
32. L. Löwenheim. Über möglichkeiten im relativkalkül. *Math. Ann.*, 76(4):447–470, 1915.
33. P. Mavridis, D. Gross-Amblard, and Z. Miklós. Using hierarchical skills for optimized task assignment in knowledge-intensive crowdsourcing. In *Proc. of WWW'16*, pages 843–853. ACM, 2016.
34. M. Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.
35. A. Nigam and N.S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
36. 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.
37. T. Sturm, M. Voigt, and C. Weidenbach. Deciding first-order satisfiability when universal and existential variables are separated. In *Proc. of LICS'16*, pages 86–95, 2016.
38. S. Tranquillini, F. Daniel, P. Kucherbaev, and F. Casati. Modeling, enacting, and integrating custom crowdsourcing processes. *TWEB*, 9(2):7:1–7:43, 2015.
39. W.M.P. Van Der Aalst, M. Dumas, F. Gottschalk, Ter H., M. La Rosa, and J. Mendling. Correctness-preserving configuration of business process models. In *Proc. of FASE'08*, pages 46–61, 2008.
40. W.M.P Van Der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, HMW Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.
41. Q. Zheng, W. Wang, Y. Yu, M. Pan, and X. Shi. Crowdsourcing complex task automatically by workflow technology. In *MiPAC'16 Workshop*, pages 17–30, 2016.

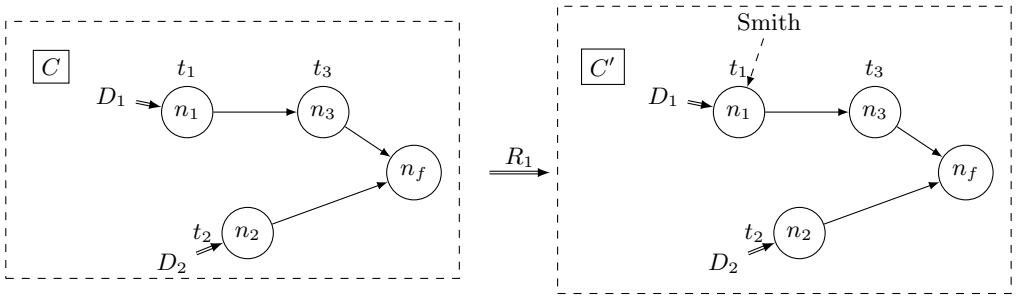

Fig. 3. Application of semantic rule R_1

A Appendix : Operational semantics of Complex Workflows

This appendix provides details on the operational semantics of complex workflows, the definition of the successor relation among symbolic configurations and the proofs of all theorems and propositions in the paper. All illustrations of semantics rules borrow examples from the SPIPOLL initiative [6].

A.1 Operational semantics

Rule 1 (Worker Assignment): A worker $u \in \mathcal{U}$ is assigned a task $t = \lambda(n)$ such that $t \notin \mathcal{T}_{aut}$. The rule applies if u is free and has the skills required by t, and if node n is not already assigned to a worker. Note that a task can be assigned to a worker even if it does not have input data yet, and is not yet executable.

$$\frac{\begin{array}{l} n \notin Dom(\mathbf{wa}) \wedge u \notin coDom(\mathbf{wa}) \wedge \lambda(n) \notin \mathcal{T}_{aut} \\ \wedge\, ((u, \lambda(n)) \in sk \vee (u, r = (\lambda(n), W_r)) \in sk) \end{array}}{(W, \mathbf{wa}, \mathbf{Dass}) \rightarrow (W, \mathbf{wa} \cup \{(n, u)\}, \mathbf{Dass})} \quad (1)$$

Consider for instance the application of rule R_1 described in Figure 3. Configurations are represented by the contents of dashed rectangles. Workflow nodes are represented by circles, tagged with a task name representing map λ. The dependencies are represented by plain arrows between nodes. Worker assignments are represented by dashed arrows from a worker name u_i to its assigned task. Data assignment are represented by double arrows from a dataset to a node. The left part of Figure 3 represents a configuration C with four nodes n_1, n_2, n_3 and n_f. The predecessors of n_1 and n_2 have been executed. Node n_1 represents occurrence of a task of type t_1 and is attached dataset D_1. Let us assume that D_1 is a database containing *bee* pictures, and that task t_1 cannot be automated ($t_1 \notin \mathcal{T}_{aut}$) and consists in tagging these pictures with bee names, which requires competences on *bee* species. Let us assume that worker *Smith* is currently not assigned any task and has competences on bees. Then $(Smith, t_1) \in sk$ and rule R_1 applies. The resulting configuration is configuration C' at the right of Figure 3, where the occurrence of t_1 represented by node n_1 is assigned to worker Smith.

Rule 2 (Atomic Task Completion): An atomic task $t = \lambda(n) \in \mathcal{T}_{ac}$ can be executed if node n is *minimal* in the workflow, 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.

$$\frac{\begin{array}{l} n \in min(W) \wedge \lambda(n) \in \mathcal{T}_{ac} \wedge \mathbf{wa}(n) = u \\ \wedge\, \mathbf{Dass}(n) \notin DB^*.\emptyset.DB^* \\ \wedge\, \exists \mathcal{D}^{out} = D_1^{out} \dots D_k^{out} \in F_{\lambda(n),u}(\mathbf{Dass}(n)), \\ \mathbf{Dass}' = \mathbf{Dass} \setminus \{(n, \mathbf{Dass}(n))\} \cup \\ \qquad \{(n_k, \mathbf{Dass}(n_k)_{[j/D_k^{out}]}) \mid n_k \in succ(n) \\ \qquad\quad \wedge n \text{ is the } j^{th} \text{ predecessor of } n_k\} \end{array}}{(W, \mathbf{wa}, \mathbf{Dass}) \xrightarrow{\lambda(n)} (W \setminus \{n\}, \mathbf{wa} \setminus \{(n,u)\}, \mathbf{Dass}')} \quad (2)$$

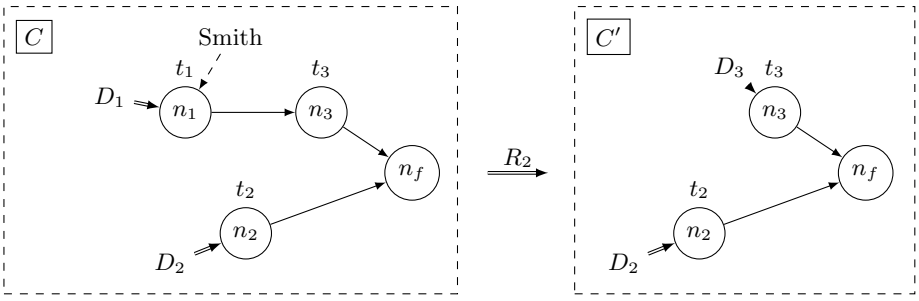

Fig. 4. Application of semantic rule R_2

Consider the example of Figure 4. We start from a configuration in which worker *Smith* has to tag bee images stored in a dataset D_1. We assume that the relational schema for D_1 is a tuple $R(id, pic)$ where id is a key and pic a picture. We also assume that tags are species names from a finite set of taxons, e.g. $Tax = \{Honeybee, Bumblebee, Masonbee, ..., Unknown\}$. Worker Smith performs the tagging task, which results in a dataset D_3 with relational schema $R'(id, pic, tag)$. One can notice that no information is given of the way worker Smith tags the pictures in D_1, the only insurance is that for every tuple $R(id, pic)$, there exists a tuple $R'(id, pic, tx)$ in D_3 where $tx \in Tax$. Notice that application of this rule may results in several successor configurations, as a human worker can choose non-deterministically any tag for each picture (including wrong answers).

Rule 3 (Automated Task Completion): An automated task $t = \lambda(n) \in \mathcal{T}_{aut}$ can be executed if node n is minimal in the workflow and its input data does not contain an empty dataset. The difference with atomic tasks completion is that $t \in \mathcal{T}_{aut}$, n is not assigned a worker, and that the produced outputs are deterministic functions of task inputs.

$$\frac{\begin{array}{l} n \in min(W) \wedge \lambda(n) \in \mathcal{T}_{aut} \wedge \mathbf{Dass}(n) \notin DB^*.\emptyset.DB^* \\ \wedge \mathcal{D}^{out} = f_{\lambda(n),u}(\mathbf{Dass}(n)) = D_1^{out} \dots D_k^{out}, \\ \mathbf{Dass}' = \mathbf{Dass} \setminus \{(n, \mathbf{Dass}(n))\} \cup \\ \quad \{(n_k, \mathbf{Dass}(n_k)_{[j/D_k^{out}]}) \mid n_k \in succ(n) \\ \quad \wedge n \text{ is the } j^{th} \text{ predecessor of } n_k\} \end{array}}{(W, \mathbf{wa}, \mathbf{Dass}) \xrightarrow{\lambda(n)} (W \setminus n, \mathbf{wa}, \mathbf{Dass}')} \quad (3)$$

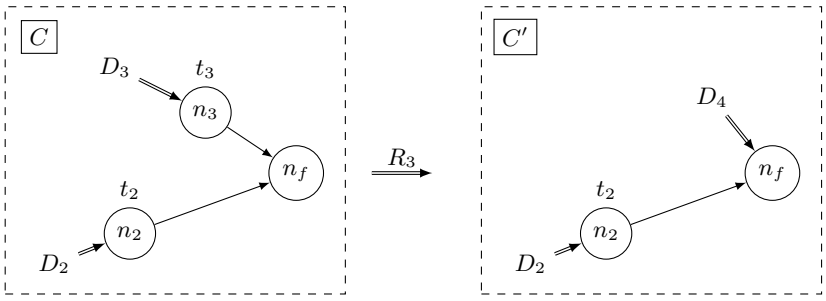

Fig. 5. Application of semantic rule R_3

Consider the example of Figure 5. We resume from the situation in Figure 4, i.e. with two nodes n_2, n_3 remaining to be executed before n_f, and with a dataset composed of tagged images attached to node n_3. Let us assume that task t_3 is an automated task that consists in pruning out images with tag "unknown". This task can be realized as a projection of D_3 of tuples $R'(id, pic, tx)$ such that $tx \neq$ "$Unknown$". As a result, we obtain a dataset D_4, used as input by node n_f such that $\forall R'(id, pic, tx) \in D_4, tx \neq$ "$unknown$" , This task can be realized by simple SQL query.

Rule 4 (Complex Task refinement): The refinement of a node n with $t = \lambda(n) \in \mathcal{T}_{cx}$ by worker $u = \mathbf{wa}(n)$ replaces node n by a workflow $W_s = (N_s, \longrightarrow_s, \lambda_s)$ if a rule $R = (t, W_s)$ exists and is listed in the competences of u. Data originally accepted as input by n are now accepted as input by the source node of W_s. All newly inserted nodes have empty input datasets.

$$\frac{\begin{array}{l} t = \lambda(n) \in \mathcal{T}_{cx} \wedge \exists u, u = \mathbf{wa}(n) \wedge (u, R) \in sk \wedge R = (t, W_S) \\ \wedge \mathbf{Dass}'(min(W_s)) = \mathbf{Dass}(n) \\ \wedge \forall x \in N_s \setminus min(W_s), \mathbf{Dass}'(x) = \emptyset^{|Pred(x)|} \\ \wedge \mathbf{wa}' = \mathbf{wa} \backslash \{(n, \mathbf{wa}(n))\} \end{array}}{(W, \mathbf{wa}, \mathbf{Dass}) \xrightarrow{ref(n)} (W_{[n/W_s]}, \mathbf{wa}', \mathbf{Dass}')} \quad (4)$$

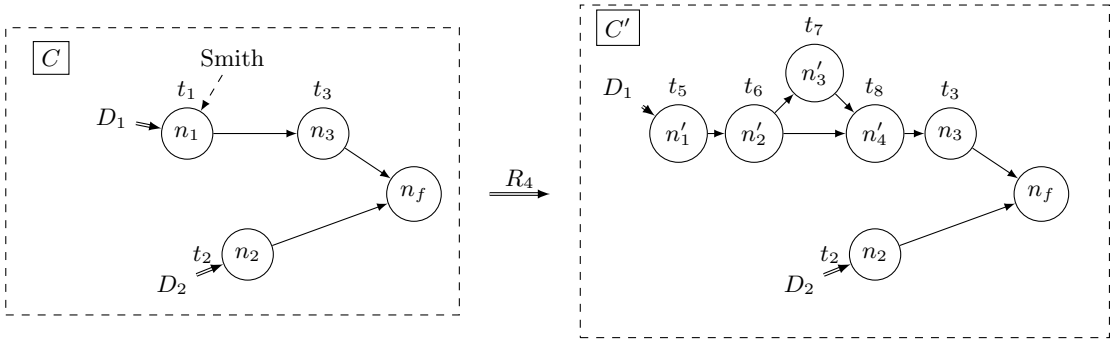

Fig. 6. Application of semantic rule R_4

Consider the example of Figure 6. Let us assume that worker "Smith" is assigned task t_1 and that this task is a complex tagging task (for instance workers are asked to find names of rare species). In such situation, Smith can decide to replace the task by a simple single-worker tagging mechanism, or by a more complex workflow, that asks a competent worker to tag pictures, then separates the obtained datasets into pictures with/without tag "Unknown", and sends the *unknown* species to an expert (for instance an entomologist) before aggregating the union of all responses. This refinement lead to a configuration C', shown in the right part of Figure 6, where n'_1 is a tagging task, n'_2 is an automated task to split a dataset, n'_3 is a tagging tasks which requires highly competent workers and n'_4 is an aggregation task. Note that conditions for worker assignment guarantee that refinement is always performed by a competent worker, owning an appropriate refinement rule to handle a task.

Note that the definition of a *complex* task is very subjective and varies from one worker to another. Classifying tasks as complex or not a priori should not be seen as a limitation, as refinement is not mandatory: a worker can replace a node n labeled by task $t \in \mathcal{T}_{cx}$ by a another node labeled by an equivalent task $t' \in \mathcal{T}_{ac} \cup \mathcal{T}_{aut}$ if this possibility is allowed by the rules she can apply. This allows to model situations where a worker has the choice to realize a task or refine it when she thinks it is too complex to be handled by a single person.

Runs of a complex workflow are successive rewritings of configurations via rules. Figure 7 gives an example of run. The top-left part of the figure is an initial con-

figuration $C_0 = (W_0, \mathbf{wa}_0, \mathbf{Dass}_0)$ composed of an initial workflow W_0, an empty map $\mathbf{wa}_0$ and a map $\mathbf{Dass}_0$ that associates dataset D_{in} to node n_i. The top-right part of the figure represents the configuration $C_1 = (W_1, \mathbf{wa}_1, \mathbf{Dass}_1)$ obtained by assigning worker u_1 for execution of task t_2 attached to node n_2 (Rule 1). The bottom part of the Figure represents the configuration C_2 obtained from C_1 when worker u_1 decides to refine task t_2 according refinement rule $(t_2, W_{t_2,1})$ (Rule 4). Workflow $W_{t_2,1}$ is the part of the Figure contained in the Grey square.

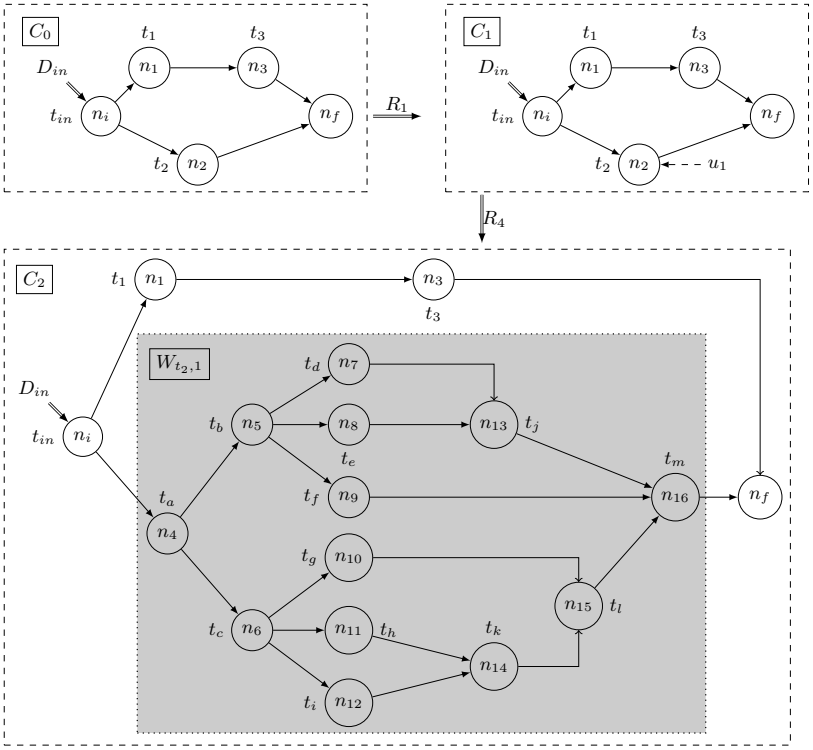

Fig. 7. Complex workflow execution. C_0 represents the initial configuration with data D_{in} allocated to node n_i. C_1 is the successor of C_0: worker u_1 is allocated to node n_2, and $t_2 = \lambda(n_2)$ is a complex task. C_3 depicts the configuration after refinement of node n_2 by a new workflow W_{t2} (shown in the Grey rectangle).

A.2 A non-terminating Complex Workflow

Let us show on an example why complex workflow may not terminate. Consider a simple linear workflow composed of three nodes : n_i, n_f, n_1 where n_1 is attached task $\lambda(n_1) = t_1$ such that $\longrightarrow = \{(n_i, n_1), (n_1, n_f)\}$. Let us assume that our system has a single worker u_1, and that this worker has the right to use rule

$rewrite_1 = (t_1, W_{t_1})$ where W_{t_1} is a workflow with three nodes w_1, w_2, w_f, such that $\lambda(w_1) = t_2$ and $\lambda(w_2) = t_1$ and where $\longrightarrow = \{(w_1, w_2), (w_1, w_f)\}$. Then, after application of semantic rule $R1$ (assigning worker u_1 to the node that carries task t_1) and semantic rule $R4$ (replacing via rewriting rule $rewrite_1$ task t_1 by W_{t_1}), one obtains a larger workflow that still contains an occurrence of task t_1. One can repeat these steps an arbitrary number of times, leading to configurations which workflow parts are growing sequences of nodes labeled by sequences of task occurrences of the form $\lambda(n_i).t_2^k.t_1.\lambda(w_f)^k.\lambda(n_f)$. In this recursive scheme, the workflow part of configurations obviously grows. Moreover, but one can easily modify this example to exhibit complex workflows with unbounded recursive schemes where the data part of configurations grows unboundedly (for instance if t_2 is a task that adds records to some dataset).

B Proofs for FO

Proposition 4. *Let X be a set of variables of the form $X = X_b \uplus X_r$ where variables from X_b take values from finite domains and variables from X_r take values in $\mathbb{R}$. Then, satisfiability of formulas of the form $\phi ::= \exists \overrightarrow{X}. \bigwedge_{i \in 1..I} R_i(X) \wedge \bigwedge_{j \in 1..J} P_j(X)$ is NP-complete.*

Proof. Let us first show that the problem belongs to NP. Let us consider an existential formula $\phi ::= \exists \overrightarrow{X}, \psi$ where ψ contains positive relational statements of the form $\phi_{R+} ::= R_1(X), \dots R_k(X)$, and negative relational statements of the form $\phi_{R-} ::= \neg R_1(X), \dots \neg R_{k'}(X)$, and predicates of the form $P_1(X), \dots P_J(X)$. For each $R_i(x_1, \dots, x_q)$ in ϕ_{R+}, with relational schema rn_i and legal domain Dom_i, we define $Ldom_i$ as the constraint $(x_1, \dots x_q) \in Dom_i$. One can choose nondeterministically in polynomial time a value d_x for each bounded variable x in X_b, and obtain a formula where only real variables appear by substitution of these variables by their value.

Then one can choose non-deterministically which relational statements and predicate hold, by guessing a truth value $v_j \in \{true, false\}$ for each relation $R_i \in 1..I$ (Resp. predicate $P_j, j \in 1..J$). Now, for each pair of choices where $rn(x_1, \dots x_q)$ holds and $rn(x'_1, \dots x'_q)$ does not, we verify that the designed tuples are disjoint, i.e. that $\neg(x_1 = x'_1 \wedge \dots \wedge x_q = x'_q)$. We call ϕ_{RxR} the formula that is the conjunction of such negations. The size of ϕ_{RxR} is in $O(r.|\phi^2|)$ where r is the maximal arity in a relational schema of the complex workflow. We can then verify that the guess of truth value for atoms yields satisfaction of ϕ, i.e. check that $\phi'_{[R_i, P_j/true, false, v_j]}$ evaluates to true. In case of positive answer, it suffices to check that with the truth value choosen for atoms, the formula $\phi_{RxR} \wedge \bigwedge_{i \in 1..k} Ldom_i \wedge \bigwedge_{v_i = true} P_i \wedge \bigwedge_{v_i = false} \neg P_i$ is satisfiable, i.e. that one can find an assignment for the real variables that are not yet valuated, which can be done in polynomial time. Now, for the hardness proof, one can easily encode a SAT problem with an FO formula over boolean variables. Checking satisfiability of a universally

quantified formula can be done in $co-NP$ in the same way, as $\forall X \phi$ is satisfiable iff $\exists X, \neg\phi$ is not. □

B.1 Proof of Proposition 1:

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 $O(r.|\psi|)$.*

Proof (Sketch). The effect of moves on the contents of datasets can be described as sequential composition of basic operations that are projections of datasets, selections, insertions of records or fields, differences, unions or joins of dataset. We first show that these basic dataset transformations allow for computation of a weakest precondition, and then show on a particular move (selection of records, Lemma 1) application of these basic properties. We do not detail all technical lemmas in this appendix, but they are rather straightforward, and will appear in an extended version of this work. Let $D_1, \dots D_k$ be the datasets that appear in ψ. If some D_i is obtained as a selection of records from a dataset D'_i, then D_i contains only records of D'_i satisfying some predicate P. The precondition will hence be obtained by a simple replacement in ψ of any statement of the form $rn(\overrightarrow{X}) \in D_i$ by $rn(\overrightarrow{X}) \in D_i \wedge P(\overrightarrow{X})$.

If D_i is obtained after insertion of a fresh record in some dataset D'_i then every statement $rn(\overrightarrow{X}) \in D_i$ can be replaced by a subformula $(rn(\overrightarrow{X}) \in D'_i \vee Dom_i(\overrightarrow{X}))$ where $Dom_i(\overrightarrow{X})$ represents constraints on legal values of inputs in a dataset with the same relational schema as D_i. Note that $Dom_i(\overrightarrow{X})$ is a quantifier free boolean combination of predicates.

Similarly, if some dataset D_i is obtained as the union of two datasets D_1, D_2, the precondition for ψ should consider that tuples that satisfy $rn(\overrightarrow{X}) \in D_i$ belong to D_1 or D_2. We simply replace atoms of the form $rs(\overrightarrow{X}) \in D_i$ by the disjunction $rn(\overrightarrow{X}) \in D_1 \vee rn(\overrightarrow{X}) \in D_2$. We also replace negative statements $rn(\overrightarrow{X}) \notin D_i$ by conjunction $rn(\overrightarrow{X}) \notin D_1 \wedge rn(\overrightarrow{X}) \notin D_2$. The size of the obtained formula is hence in $O(2.|\psi|)$.

If some D_i is obtained by creation of a new field for each record in dataset D'_i, then relational statement $rn(\overrightarrow{X}) \in D_i$ is replaced by another statement $rn(\overrightarrow{Y}) \in D'_i \wedge P'(\overrightarrow{Y})$ where $\overrightarrow{Y}$ is a subset of $\overrightarrow{X}$, and $P'(\overrightarrow{Y})$ is a quantifier free predicate indicating constraint on $\overrightarrow{Y}$ obtained after variable elimination when $\overrightarrow{X}$ takes legal values imposed by relational schema of D_i (i.e. it satisfies $Ldom_i$ –see proof of Proposition 4–) and satisfies the constraints of relational schema of D'_i. As we assume that arithmetic predicates are simple (two-dimensional) inequalities, the size of P' is not larger than that of $Ldom_i$. The weakest precondition can hence double the number of atoms, but keeps the same bound on the number of variables used.

For (binary) joins, relation of the form $rs(\overrightarrow{X}_i)$ are transformed in statements of the form $rs(\overrightarrow{Y_i}) \wedge rs(\overrightarrow{Z_i}) \wedge y_1 = z_1$, where $y_1 \in \overrightarrow{Y_i}$ and $z_1 \in \overrightarrow{Z_i}$. As every relational statement can be replaced and hence create new variables, this calculus of a weakest precondition may give a formula of size in $O(2.|\psi|)$.

Last, if some D_i is a projection of some dataset D'_i on a subset of its field, then the weakest precondition for ψ may multiply the number of variables in use by r, whence giving a formula of size in $O(r.|\psi|)$.

We rely on technical lemmas that detail the construction of a weakest precondition for each particular type of move.

B.2 A technical lemma for weakest precondition

The computation of weakest precondition can be effectively performed for every type of task (automated execution of SQL requests, inserttion/creation/deletion of answers by workers, refinement,...). We give below an example of weakest precondition calculus for a task that performs a selection of records that satisfy some predicate in a dataset. For all other types of actions, computing a weakest precondition follows the same lines. For conciseness, we do not detail all of them. The whole set of lemma for each type of action is considered in an extended version of this work.

Lemma 1 (Weakest precondition for Selection of records). *Let φ be a FO formula, and mv be a move that selects records that satisfy a predicate P from datasets. Then one can effectively compute an FO formula $\psi = wp[mv]\varphi$. Moreover, if φ is in $\forall$FO (resp. $\exists$FO, BSR, SF) and P is an arithmetic/boolean predicate then ψ is also in $\forall$FO (resp. $\exists$FO, BSR, SF).*

Proof. Let $D'_1, \dots D'_j, \dots D'_k \models \varphi$, and let D'_j be a dataset with relational schema $rs = (rn, A)$ obtained by selection of records from an input dataset D_i with relational schema $rs(rn, A)$. One can notice that selection keeps the same relational schema, and in particular the same set of attributes $A = (a_1, \dots a_k)$. We will assume that selected records are records that satisfy some predicate $P(v_1, \dots v_k)$ that constrain the values of a record (but do not address properties of two or more records of D_i). That is, the records selected from D_i by P are records that satisfy $\psi_{sel} = \exists v_1, \dots v_k, rn(v_1, \dots v_p) \wedge P(v_1, \dots, v_k)$. We want to compute $\psi = wp[Selection(\psi_{sel})]\ \varphi$.

Formula φ is a formula of the form $\alpha(\overrightarrow{X}).\phi$, where $\alpha(\overrightarrow{X})$ is a prefix. It contains K_{rn} subformulas of the form $rn(w_i, \dots w_{i+k})$ or $\neg rn(w_i, \dots, w_{i+k})$ and we assume without loss of generality that these subformulas are over disjoint sets of variables (one can add new variables and equalities if this is not the case). Let $\phi_{rn,1}, \dots \phi_{rn,K_{rn}}$ be the subformulas of ϕ addressing tuples with relational schema rs. For $i \in 1..K_{rn}$, we let $\phi^P_{rn,i}$ denote the formula $rn(w_i, \dots w_{i+k}) \wedge P(w_i, \dots w_{i+k})$ if $\phi_{rn,i}$ is in positive form and $\neg(rn(w_i, \dots w_{i+k}) \wedge P(w_i, \dots w_{i+k}))$ otherwise. Let us denote by $\phi_{[\{\varphi_{rn,i}\}|\{\varphi^P_{rn,i}\}]}$ the formula ϕ where every $\phi_{rn,i}$ is replaced by $\phi^P_{rn,i}$.

The weakest precondition on $D'_1, \dots D_i, \dots, D'_k$ for a selection operation with predicate P is defined as

$$\psi = wp[Selection(\psi_{sel})]\varphi = \alpha(\overrightarrow{X}).\phi_{[\{phi_{rn,i}\}|\{phi^P_{rn,i}\}]}$$

Last, one can notice that transforming $\alpha(\overrightarrow{X}).\phi$ into $\alpha(\overrightarrow{X}).\phi_{[\{phi_{rn,i}\}|\{phi^P_{rn,i}\}]}$ does not introduce new variables, and preserve the prefix of the formula. As φ and ψ start with the same prefix $\alpha(\overrightarrow{X})$, we can claim that φ and ψ are in the same fragment of FO. □

One can notice that this weakest precondition is a rather syntactic transformation, that replaces atoms of the form $rn(x_1, \dots x_k)$ by $rn(x_1, \dots x_k) \wedge P(x_1, \dots x_k)$. If $x_1, \dots, x_k$ are all existentially quantified variables (resp. all universally quantified variables in φ, then they remain existentially (resp universally quantified) in ψ. Hence, if φ is in ∃FO,∀FO,BSR, SF-FO then so is $wp[Selection(\psi_{sel})]\varphi$. The same remark applies to all other types of moves (we do not give proofs for each move here, but they follow the same line as for selection, and will be available in an extended version of this work).

C Appendix : proofs for termination

C.1 Proof of Theorem 1

Theorem 1 Existential termination of complex workflows is an undecidable problem.

Proof. The proof is done by reduction from the halting problem of two counter machines to termination of complex workflows. A 2-counter machine (2CM) is a tuple $\langle Q, c_1, c_2, I, q_0, q_f \rangle$ where:

- Q is a finite set of states.
- $q_0 \in Q$ is the initial state, $q_f \in Q$ is a particular state called the final state.
- c_1, c_2 are two counters holding non-negative integers.
- $I = I_1 \cup I_2$ is a set of instructions. Instructions in I_1 are of the form $inst_q = inc(q, c_l, q')$, depicting the fact that the machine is in state q, increases the value of counter c_l by 1, and moves to a new state q'. Instructions in I_2 are of the form $inst_q = dec(q, c_l, q', q'')$, depicting the fact that the machine is in state q, if $c_l == 0$, the machine moves to new state q' without making any change in the value of counter c_l, and otherwise, decrements the counter c_l and moves to state q''. We consider deterministic machines, i.e. there is at most one instruction $inst_q$ per state in $I_1 \cup I_2$. At any instant, the machine is in a configuration $C = (q, v_1, v_2)$ where q is the current state, v_1 the value of counter c_1 and v_2 the value of counter c_2. The machine executes instructions from its current configuration, and stops as soon as it reaches state q_f.

From a given configuration $C = (q, v_1, v_2)$, a machine can only execute instruction $inst_q$, and hence the next configuration $\Delta(C)$ of the machine is also unique. A run of a two counters machine is a sequence of configurations $\rho = C_0.C_1 \dots C_k$ such that $C_i = \Delta(C_{i-1})$. The halting problem is defined as follows: given a 2-CM, an initial configuration $C_0 = (q_0, 0, 0)$, decide whether a run of the machine reaches some configuration (q_f, n_1, n_2), where q_f is the final state and n_1, n_2 are arbitrary values of the counter. It is well known that this halting problem is undecidable.

Let us now show how to encode a counter machine with complex workflows.

- We consider a dataset D with relational schema $rs = (R, \{k, cname\})$ where k is a unique identifier, and $cname \in Cnt_1, Cnt_2, \perp$. Clearly, we can encode the value of counter c_x with the cardinal of $\{(k, n) \in D \mid n = Cnt_x\}$. We start from a configuration where the dataset contains a single record $R(0, \perp)$
- For every instruction of the form $inc(q, c_x, q')$ we create a task t_q, and a workflow W_q^{inc}, and a worker u_q, who is the only worker allowed to execute these tasks. The only operation that u_q can do is refine t_q with workflow W_q^{inc}. W_q^{inc} has two nodes n_q^{inc} and $n_{q'}$ such that $(n_q^{inc}, n_{q'}) \in \longrightarrow$, $\lambda(n_q^{inc}) = t_q^{inc}$ and $\lambda(n_{q'}) = t_{q'}$. Task t_q^{inc} is an atomic task that adds one record of the form (k', Cnt_x) to the dataset. Hence, after executing tasks t_q and t_q^{inc}, the number of occurrences of Cnt_x has increased by one.
- For every instruction of the form $dec(q, c_x, q', q'')$, we create a complex task t_q and a worker u_q who can choose to refine t_q according to rule $(t_q, W_{q,Z})$ or rule $(t_q, , W_{q,NZ})$. The choice of one workflow or another will simulate the decision to perform a zero test or a non-zero test. Note that as the choice of a particular rule to replace a task workflow is non-deterministic, worker u_q can choose one or the other.
- Let us now detail the contents of $W_{q,NZ}$, represented in Figure 8. This workflow is composed of nodes $n_q^{div}, n_q^{C_x}, n_q^{C_{\bar{x}} \cup \perp}, n_q^{\otimes}, n_q^{dec}$ and $n_{q''}$, respectively labeled by tasks $t_q^{div}, t_q^{C_x}, t_q^{C_{\bar{x}} \cup \perp}, t_q^{\otimes}$, t_q^{dec} and $t_{q''}$. The dependence relation in $W_{q,NZ}$ contains pairs $(n_q^{div}, n_q^{C_x})$, $(n_q^{div}, n_q^{C_{\bar{x}} \cup \perp})$, $(n_q^{C_x}, n_q^{\otimes})$, $(n_q^{C_{\bar{x}} \cup \perp}, n_q^{\otimes})$, $(n_q^{\otimes}, n_q^{dec})$ and $(n_q^{dec}, n_{q'})$. The role of t_q^{div} is to split $\mathbf{Dass}(n_q^{div})$ into disjoint parts: the first one contains records of the form $R(k, C_x)$ and the second part consists of all other remaining records. Tasks $t_q^{C_x}$ and $t_q^{C_{\bar{x}} \cup \perp}$ simply forward their inputs, and task $t_q^{\otimes}$ computes the union of its inputs. Note however that if one of the inputs is empty, the task cannot be executed. Then, task t_q^{dec} deletes one record of the form $R(k, C_x)$. Hence, if $D_q = \mathbf{Dass}(n_q)$ is a dataset that contains at least one record of the from $R(k, C_x)$, the execution of all tasks in $W_{q,NZ}$ leaves the system in a configuration with a minimal node $n_{q''}$ labeled by task $t_{q''}$, and with $\mathbf{Dass}(n_{q'}) = D_q \setminus R(k, C_x)$
- Let us now detail the contents of $W_{q,Z}$. This workflow is composed of nodes n_q^{div},$n_q^{C_x \cup \perp}$, n_q^{id}, n_q^{btest}, n_q^{done}, $n_{q'}$ respectively labeled by tasks t_q^{div},$t_q^{C_x \cup \perp}$, t_q^{id}, t_q^{btest}, t_q^{done}, t'_q. The flow relation if given by pairs $(n_q^{div}, n_q^{C_x \cup \perp})$, (n_q^{div}, n_q^{id}) $(n_q^{C_x \cup \perp}, n_q^{btest})(n_q^{btest}, n_q^{done})$ and (n_q^{id}, n_q^{done}). The role of task t_q^{div} is to project its input dataset on records with $cname = C_x$ or $cname = \perp$, and forwards the obtained dataset to node $n_q^{C_x \cup \perp}$. On the other hand, it creates a copy of

the input dataset and forwards it to node n_q^{id}. The role of task $t_q^{C_x \cup \perp}$ is to perform a boolean query that returns $\{true\}$ if the dataset contains a record $R(k, C_x)$ and $\{false\}$ otherwise, and forwards the result to node n_q^{btest}. Task t_q^{btest} selects records with value $\{false\}$ (it hence returns an empty dataset as the result of the boolean test was $\{true\}$). Task t_q^{id} forwards its input to node n_q^{done}. Task t_q^{done} received input datasets from n_q^{btest} and n_q^{id} and forwards the input from n_q^{id} to node $n_{q''}$. One can immediately see that if the dataset input to n_q^{div} contains an occurrence of C_x then one of the inputs to n_q^{done} is empty and hence the workflow deadlocks. Conversely, if this input contains no occurrence of C_x, then this workflows reached a configuration with a single node $n_{q''}$ labeled by task $t_{q''}$, and with the same input dataset as n_q.

One can see that for every run $\rho = C_0 \dots C_k$ of the two counter machine, where $C_k = (q, v_1, v_2)$ there exists a single non-deadlocked run of the complex workflow, that terminates of configuration $(W, \mathbf{wa}, \mathbf{Dass})$ where W consists of a single node n_q labeled by task t_q, and such that $\mathbf{Dass}(n_q)$ contains v_1 occurrences of records of the form $R(k, C_1)$ and v_2 occurrences of records of the form $R(k, C_2)$. Hence, a two counter machine terminates in a configuration (q_f, v_1, v_2) iff the only non-deadlocked run of the complex workflow that encodes the two counter machine reaches a final configuration.

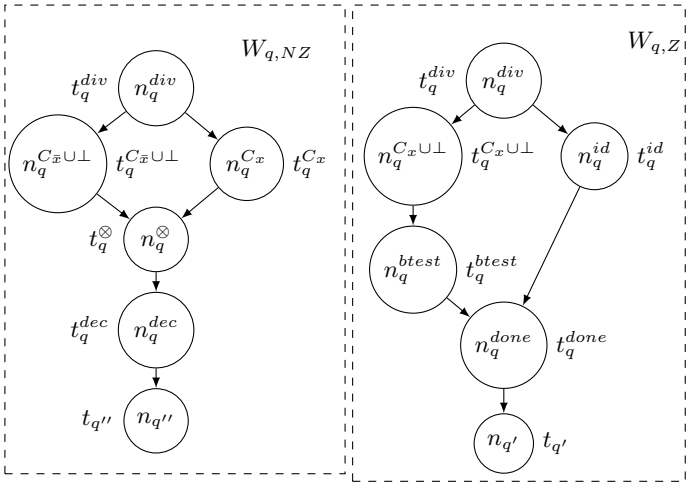

Fig. 8. Encoding of Non-zero test followed by decrement (left), and Zero Test followed by state change (right).

C.2 Successor relation among symbolic configurations

From a symbolic configuration, that does not describe the exact contents of datasets in use, we can compute the effects of application of a particular rule, i.e. compute symbolic descriptions of configurations that appear after a move. A symbolic configuration $C_j^S = (W_j, \mathbf{wa}_j, Dass_j^S)$ is the *successor* of a symbolic configuration $C_i^S = (W_i, \mathbf{wa}_i, Dass_i^S)$ iff one of the following situation holds:

- there exists a worker $u \in \mathcal{U}$ and a node $n \in W_i$ such that $\mathbf{wa}_i^{-1}(u) = \emptyset$, $\mathbf{wa}_i(n) = \emptyset$, $\exists(u, \lambda(n)) \in sk$ or $(u, r) = (\lambda(n), W_t)) \in sk$, and $W_j = W_i$, $\mathbf{Dass}^S = \mathbf{Dass}^S$ and $\mathbf{wa}_j = \mathbf{wa}_i \uplus \{(n, u)\}$.
- there exists $n \in min(W_i)$ such that $t = \lambda(n)$ is an automated task manipulating datasets $D_1, \ldots D_q$, n has k successors $n_1, \ldots n_k$, $W_j = W_i \setminus \{n\}$, $\mathbf{Dass}^S$ assigns to each successor $n_j, j \in 1..k$ the relational schema rs_j^{out}, and $\mathbf{wa}_j = \mathbf{wa}_i$.
- there exists $n \in min(W_i)$ such that $t = \lambda(n)$ is an atomic task manipulating datasets $D_1, \ldots D_q$, with and n has k successors $n_1, \ldots n_k$, $W_j = W_i \setminus \{n\}$, $\mathbf{Dass}^S$ assigns to each successor $n_j, j \in 1..k$ the relational schema rs_j^{out}, and $\mathbf{wa}_j = \mathbf{wa}_i \setminus \{(n, \mathbf{wa}_i(n))\}$.
- there exists $n \in W_i$, $\lambda(n)$ is a complex task, and $\mathbf{wa}(n) = u$, W_j is the workflow obtained by replacement of n in W_i by a workflow W^{new} such that $r = (\lambda(n), W^{new}) \in \mathcal{R}$, and $(u, r) \in sk$. $\mathbf{Dass}^S$ assigns to the copy of minimal node n_j of W^{new} the relational schemas in $Dass_i^S(n)$, and $\mathbf{wa}_j = \mathbf{wa}_i \setminus \{(n, u)\}$.

C.3 Proof of Proposition 2

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 $\rho^S = C_0^S \ldots C_i^S$ 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*

Proof. We have to check feasibility of ρ^S, that is starting from C_i^S, check that all conditions met along a run with signature ρ^S to reach a configuration C_i compatible with C_i^S are met at each step, and allow for existence of configurations $C_0, C_1, \ldots C_{i-1}$. First notice that the actual run with signature ρ^S performs the same sequence of moves as in ρ^S, and that the question of existence of a run ρ with signature ρ^S only needs to verify satisfiability of constraints on data computed at each step of this run, not the sequence of moves along ρ. Second, one can notice that if ρ^S contains a deadlock, it is necessarily the last symbolic configuration of the run, as for every symbolic configuration from C_0^S up to C_{i-1}^S we are able to find a successor configuration. So one needs not check existence of a deadlock separately when checking feasibility of ρ^S, and we mainly have to check for emptiness

of datasets for configurations that are potential deadlocks. A third remark is that semantic rules that affect workers to tasks or perform a refinement do not consider data contents. Hence, if the move from C_{i-1} to C_i is a worker assignment or a refinement, then it is necessarily feasible as long as C_{i-1} is reachable. The only cases where data can affect execution of a step along a run is when an automated task or an atomic task has to process empty data. For each of these steps, one has to check that the inputs of an executed task t are not empty, i.e. suppose that $D_1 \neq \emptyset \wedge \cdots \wedge D_k \neq \emptyset$ for some datasets $D_1 \dots D_k$ used by t. Non-emptiness of a dataset D_k is simply encoded by the $\exists$FO formula $\exists \overrightarrow{X}, rn(\overrightarrow{X}) \in D_k$.

Non-emptiness of a dataset D_k at some configuration C_j is a property that depends on properties of previous steps in the execution. For instance, if the move from C_{j-1} to C_j realizes the projection of a dataset, i.e. filters records in a dataset D'_k to keep only those that satisfy some predicate P, then the precondition that must hold at C_{j-1} is $\psi_{j-1} ::= \exists \overrightarrow{X}, rn(\overrightarrow{X}) \wedge P(\overrightarrow{X})$. We have seen in Proposition 1 that , if the move $C_{j-1} \xrightarrow{m_{i-1}} C_{i-1}$ is a transformation of records, a transformation of some dataset that adds new fields, a join of two datasets, the formula ψ_{j-1} is effectively computable. Further, from Corollary 1, if tasks do not use SQL difference, the weakest precondition of an existential FO formula is an existential FO formula.

This generalizes to the whole signature. Let ψ_k be an $\exists$FO formula that has to be satisfied by configuration C_k in a run compatible with signature ρ^S. There exists a sequence of moves $C_0 \xrightarrow{m_1} C_1 \dots \xrightarrow{m_k} C_k$ iff the sequence $C_0 \xrightarrow{m_1} C_1 \dots \xrightarrow{m_{k-1}} C_{k-1}$ ends in a configuration C_{k-1} such that $C_{k-1} \models wp[m_k]\psi_k$ (by definition of weakest precondition). One can decide whether $wp[m_k]\phi_k$ is satisfiable, as ψ_k is in $\exists$FO, and by Prop. 1, $wp[m_k]\psi_k$ is effectively computable and in the $\exists$FO fragment. If $wp[m_k]\psi_k$ is not satisfiable, then the move from C_{k-1} to C_k *always* ends with datasets that do not fulfill ψ_k. If $wp[m_k]\psi_k$ is satisfiable, then the runs that reach C_{k-1} are realizable only if we assume that several datasets (used as input of some task realized at step k) are non-empty at stage $k-1$. We then have to add statements of the form $D_i \neq \emptyset$ to obtain a formula that should hold at step $k-1$, and get a formula of the form $\psi_{k-1} ::= wp[m_k]\psi_k \wedge D_1 \neq \emptyset \wedge \cdots \wedge D_m \neq \emptyset$. This adds only an existential conjunction, so ψ_{k-1} is also in $\exists$FO.

Now, one can starts from $\psi_i ::= true$ and build inductively all weakest preconditions $\psi_{i-1}, \psi_{i-2}, \dots, \psi_0$ that have to be satisfied respectively by configurations $C_{i-1}, \dots, C_0$ so that an actual run of the complex workflow with signature $C_0^S \dots C_i^S$ exists. If any of these preconditions is unsatisfiable, then there exists no run with signature $C_0^S \dots C_k^S$ leading to a configuration C_i compatible with C_i^S, and hence ρ^S is not the signature of an actual run of CW. The size of ψ_0 is in $O(i.r^i)$. Indeed, we add obligations to prove non-emptiness at each step k, but proving satisfiability of $\exists \overrightarrow{X}, \phi(\overrightarrow{X}) \wedge \exists \overrightarrow{X}, \phi(\overrightarrow{X})$ amounts to checking separately satisfiability of $\exists \overrightarrow{X}, \phi(\overrightarrow{X})$ and $\exists \overrightarrow{Y}, \phi(\overrightarrow{Y})$. According to Prop. 1, the size of $wp[m_k]\psi$ is in $O(r.|\psi|$, where r is the maximal arity of relational schemas of the complex workflow. So one can check separately satisfiability of $\exists \overrightarrow{X}, \phi(\overrightarrow{X})$ and $D_x \neq \emptyset$, and

maintain a series of $O(i)$ formulas of total size in $O(i.r^i)$. Hence, as ψ_0 is still in the existential fragment of FO, and as checking satisfiability of an existential FO formula is in $EXPTIME$ in the size of the formula (by proposition 4), checking all preconditions for a run of size k compatible with ρ^S can have a complexity that is in $O(2^{r^i})$.

Assume that $\psi_{k-1}, \psi_{k-2}, \dots \psi_0$ are satisfiable. It remains to show that the input(s) of the complex workflow satisfy the weakest precondition for the execution of ρ^S, i.e. satisfy ψ_0. Then, when the input is a single dataset D_{in}, it remains to check that $D_{in} \models \psi_0$ to guarantee existence of a run with signature $C_0^S \dots C_i^S$ that starts with input data D_{in} and leads to a configuration C_k. This is a standard model checking question, which can be solved in $O(|D_{in}|^{|\psi_0|})$, that is in $O(|D_{in}|^{r^k})$. As the complexity of checking satisfiability of weakest preconditions $\psi_k \dots \psi_0$ is already in $2EXPTIME$, the overall complexity is in $2EXPTIME$.

Similarly, if $\mathcal{D}_{in}$ is given as an FO formula, the complexity depends on the considered fragment used to specify $\mathcal{D}_{in}$. In general, if $\mathcal{D}_{in}$ is given as an FO formula, it is undecidable if $\mathcal{D}_{in} \wedge \psi_0$ is satisfiable. If $\mathcal{D}_{in}$ is an existential formula, then the complexity is exponential in the size of $\mathcal{D}_{in}$ and also exponential in the size of ψ_0. Assuming that $|\mathcal{D}_{in}| \leq 2^k$ we have a $2EXPTIME$ complexity. If $\mathcal{D}_{in}$ is in the BSR fragment, then checking satisfiability of $\mathcal{D}_{in}$ is in $NEXPTIME$ [28], and so the overall complexity needed to check existence of a run with signature ρ^S from a dataset in $\mathcal{D}_{in}$ is in 2EXPTIME w.r.t the size of $\mathcal{D}_{in} \wedge \psi_0$, and hence 3EXPTIME. If $\mathcal{D}_{in}$ is a universal formula then we can use standard mini-scoping rules to transform $\mathcal{D}_{in} \wedge \psi_0$ is a formula in the BSR fragment, yielding again a 3EXPTIME complexity. Last if $\mathcal{D}_{in}$ is in the separated fragment of FO, then checking its satisfiability is n_{in}-fold exponential in the size of the formula depicting $\mathcal{D}_{in}$, so the overall process of checking realizability of ρ^S has an n_{in}-fold exponential complexity. □

C.4 Recursion freeness: decidability an bounds on runs

Proposition 5. *Let $CW = (W_0, \mathcal{T}, \mathcal{U}, sk, \mathcal{R})$ be a complex workflow. One can decide if CW is recursion free in $O(|\mathcal{T}_{cx}^2| + |\mathcal{R}|)$.*

Proof. Building $RG(CW)$ can be done in $O(|\mathcal{R}|)$. Checking existence of a cycle in $RG(CW)$ that is accessible from some task in W_0 can be done in polynomial time in the size of $RG(CW)$, for instance using a DFS algorithm, than runs in time in $O(|\mathcal{T}_{cx}|^2)$. □

In executions of recursion free CWs, a particular task t can be replaced by a workflow that contains several tasks $t_1, \dots t_k$ that differ from t. Then, each t_i can be replaced by workflows combining other tasks that are not t nor t_i, and so on... For simplicity, we assume that W_0 and all workflows in rules have nodes labeled by distinct task names. We can then easily prove the following property:

Proposition 6. *Let $C = (W, \mathbf{wa}, \mathbf{Dass})$ be a configuration of a recursion free complex workflow CW. Then there exists a bound $K_{\mathcal{T}_{cx}}$ on the size of W, and the length of a (symbolic) execution of CW is in $O(3.K_{\mathcal{T}_{cx}})$*

Proof (Sketch). We assume, without loss of generality, that all workflows in all rules have nodes labeled by distinct task names, and that the initial workflow has a single node. Let d be the maximal number of new occurrences of complex tasks that can be rewritten in one refinement (i.e., the maximal number of complex tasks that appear in a rule). Each rewriting adds at most $d-1$ complex tasks to the current configuration. The number or rewritings is bounded, as CW is recursion free. For a given node n appearing in a configuration C_k along a run, one can trace the sequence of rewritings $Past(n)$ performed to produce n. According to recursion freeness, when a node n is replaced by a workflow W_t, then none of the tasks labeling nodes of W_t appears in $Past(n)$. Hence, the number of nodes in a configuration is at most $K_{\mathcal{T}_{cx}} = d^{\mathcal{T}_{cx}}$. Now, for a given configuration, the number of applications of semantic rules $R1$,$R2$, and $R3$ is bounded, and decreases the number of nodes in the workflow part of the configuration. □

C.5 Proof of Proposition 3

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.

Proof. Let us first prove that if *i*) fails, a complex workflow does not terminate universally. If CW has recursive task rewriting, then there is a cycle in the rewriting graph $RG(CW)$ that is accessible from a task $t_0 = \lambda(n_0)$ appearing in W_0. Hence, there is an infinite run $\rho^\infty = C_0 \xrightarrow{a_1} C_1 \xrightarrow{r_1} C_2 \dots$ of CW which moves are only worker assignments (moves of the form a_i in ρ^∞) to a node of the current workflow at configuration C_i followed by a rewriting (moves of the form r_i in ρ^∞) that creates new instances of tasks, such that the sequence of rewritten task follows the same order as in the cycle of $RG(CW)$. Similarly, if CW terminates universally, then all runs are finite, and infinite runs of the form ρ^∞ cannot exist, and CW must be recursion free.

We can now address point *ii*) If CW can reach a deadlocked configuration, then by definition, it does not terminate. If all runs of CW terminate, then from any configuration, there is a way to reach a final configuration, and hence no deadlock is reachable.

The need for the last point *iii*) is proved in lemma 2 below. □

Lemma 2. *Let C_i^S be a potential deadlock with successors $C_{i,1}^S, \dots C_{i,k}^S$ corresponding respectively to execution of tasks attached to minimal nodes $n_1, \dots n_k$ in the workflow part of node C_i^S. Then a run ρ with signature $\rho^S = C_0^S \dots C_i^S$ such that $D_k = \emptyset$ for some $D_k \in \mathbf{Dass}(n_j)$ does not terminate.*

Proof. If a node n_j in a configuration C_i with signature C_i^S is labeled by a task and is attached an empty dataset, then any sequence of worker assignment, refinement, or task execution occurring from C_i will result in a configuration C_i' where

either n_j is still attached an empty dataset, or n_j was replaced, but the refinement produced fresh nodes with an empty dataset attached to it. Then either n_j or one of its refinements will never be executed, and the workflow cannot reach a final configuration.□

This lemma has useful consequences: it is sufficient to detect a run with signature $C_0^S \dots C_i^S$ as prefix, where $C_i^S = (W_i, \mathbf{wa}_i, \mathbf{Dass}_i^S)$, and to prove that a node n_j in W_i can have an empty input dataset D_{n_j} to claim that there exists an execution that deadlocks in CW.

C.6 Proof of Theorem 2

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
- $co - n_{in}$-fold-$EXPTIME$, where $n_{in} = |\mathcal{D}_{in}| + 2^K$ if $\mathcal{D}_{in}$ is in SF-FO.

Proof. Complex workflows terminate iff they have bounded recursive schemes, and if they do not deadlock. Condition i) can be verified in $O(|\mathcal{T}_{cx}^2| + |\mathcal{R}|)$ (see proposition 5).

If CW is recursion free, then condition ii) can be verified non-deterministically by guessing a symbolic execution $C_0^S \dots C_k^S$ of length at most $3.K_{\mathcal{T}_{cx}}$. If this execution deadlocks, then there is an execution of CW that does not terminate, and we can safely conclude that CW does not terminate universally (for any input). Absence of deadlocks can hence be checked in $EXPTIME$.

If this execution contains a potential deadlock at symbolic configuration C_i^S, then C_i is a configuration from which a particular dataset D must be used as input by a task, and must be empty to cause a deadlock. Before concluding that C_i^S can be a real deadlock, one has to check whether there exists an actual real execution $C_0 \dots C_i$ such that property $D = \emptyset$ holds at C_i. Emptyness of D can be encoded as by a universal FO formula of the form $\psi_i ::= \forall \overrightarrow{X}, rn(\overrightarrow{X}) \notin D$. We can show that the precondition ψ_{i-1} that needs to hold at C_{i-1} in order to obtain an empty dataset D at step C_i are still of the form $D' = \emptyset$ (hence expressible via an universal formula) for some D' (this is the case if the move from C_{i-1} to C_i just adds a field to D' to obtain D) or an FO formula in the universal fragment computed as the weakest precondition $wp[m_i]\psi_i$. Then one can repeat the following steps at each step $k \in i-1, i-2, \dots$ up to C_0^S:

- compute $\psi_k = wp[m_k]\psi_{k+1}$. We know that this weakest precondition can be computed, and that ψ_k is still an universal formula of size in $O(r.|\psi_{k+1}|)$ (see proposition 1).

– Check satisfiability of ψ_k. If the answer is false, then Fail: one cannot satisfy the conditions required to have $D = \emptyset$ at step i, and hence there is no execution with signature $C_0^S \dots C_i^S$ that deadlocks at C_i, the randomly chosen execution is not a witness for deadlock. If the answer is true, continue.

If the algorithm does not stop before step $k = 0$, then the iteration computes a satisfiable formula ψ_0 of size in $O(r^i)$. It remain to show that inputs of the complex workflow meet the conditions in ψ_0.

Let us assume that the universal termination question is considered for a single input dataset D_{in}, one has to check that $D_{in} \models \psi_0$. As ψ_0 is a universal formula i.e. is of the from $\forall \overrightarrow{X}, \varphi_0$, this can be done in $O(|D_{in}|^{|\psi_0|})$. If the answer is true then we have found preconditions that are satisfied by D_{in} and that are sufficient to obtain an empty dataset in at configuration C_i in a run $C_0 \dots C_i$ that has signature $C_0^S \dots C_i^S$, i.e. $C_0^S \dots C_i^S$ witnesses the existence of a deadlock. Overall, one has to solve up to $i < 3.K_{\mathcal{T}_{cx}}$ satisfiability problems for universal FO formulas $\psi_{i-1}, \psi_{i-2} \dots \psi_1$ of size smaller than r^i, and a model checking problem for input D_{in} with a cost in $O(|D_{in}|^{r^i})$. The satisfiability problems are $NEXPTIME$ in the size of the formula [28] whence checking satisfiability of $\psi_{i-1}, \dots, \psi_0$ has a complexity that is doubly exponential in $K_{\mathcal{T}_{cx}}$. Considering that data fields are encoded with c bits of information, D_{in} is a dataset of size in $O(2^{r.c})$. Hence, the overall complexity to check that $C_0^S \dots C_i^S$ is a witness path that deadlocks is in $2 - EXPTIME$.

Conversely, if the universal termination question is considered for a several input datasets described with an FO formula $\mathcal{D}_{in}$, one has to check that no contradiction arises when requiring existence of an input D_{in} that satisfies both $\mathcal{D}_{in}$ and ψ_0. This can be done by checking the conjunction $\mathcal{D}_{in} \wedge \psi_0$. Now, this formula is of size in $|\mathcal{D}_{in}| + |\psi_0|$, and one can consider variables in $\mathcal{D}_{in}$ and ψ_0 to be disjoint. Standard equivalence rules (miniscoping rules, see [37]) allow to rewrite this conjunction into an equivalent formula in prenex normal form. If $\mathcal{D}_{in}$ is in $\forall$FO, then $\mathcal{D}_{in} \wedge \psi_0$ is in $\forall$FO and we have a co-2EXPTIME procedure to verify its satisfiability. For fragments ($\exists$FO, BSR-FO), $\mathcal{D}_{in} \wedge \psi_0$ fall is the class of BSR-FO or SF-FO formulas, but with three alternations, with an NEXPTIME complexity of satisfiability, yielding a triple exponential complexity. For $\mathcal{D}_{in}$ in SF-FO, the complexity of the last step is n_{in}-fold exponential in the size of $\mathcal{D}_{in}+$ the size of ψ_0. As for the unique input case, if $\mathcal{D}_{in} \wedge \psi_0$ is satisfiable, then $C_0^S \dots C_i^S$ witnesses existence of a non-terminating execution.

Hence, one can witness existence of a non-terminating run in $O(K_{\mathcal{T}_{cx}}.2^{r^{K_{\mathcal{T}_{cx}}}} + C_{in})$ where C_{in} is the cost required to check satisfiability of $\mathcal{D}_{in} \wedge \psi_0$.

Last if $\mathcal{D}_{in}$ is specified in an undecidable fragment of FO, then one cannot conclude whether there exists a legal input that satisfies $\mathcal{D}_{in}$ and ψ_0.

D Proofs for Section 6

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 of FO.*
– *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 of FO.*

Proof. Let us first prove the undecidability part: It is well known that satisfiability of FO is undecidable in general, and in particular for the AE fragment with formulas of the form $\forall \vec{X} \exists \vec{Y}, \phi(X, Y)$. Hence $\psi_{AE}^{in,out}$ can be a formula which satisfiability is not decidable. One can take an example of formula ψ_{unsat} which satisfiability is not decidable. One can also build a formula ψ_{id} that says that the input and output of a workflow are the same. One can design a workflow CW_{id} with a single final node which role is to return the input data, and set as client constraint $\psi^{in,out} = \psi_{unsat} \wedge \psi_{id}$. This workflow has a single run. Then, CW_{id} terminates properly iff there exists a dataset D_{in} such that $D_{in} \models \psi_{unsat}$, i.e. if ψ_{unsat} is satisfiable. Universal and existential proper termination are hence undecidable problems.

For the decidable cases, one can apply the technique of Theorem 2. One can find non-determinisitically a symbolic run ρ^S that does not terminate and check that it is the signature of an actual run, or a symbolic run ρ^S that terminates and check whether it satisfies $\psi^{in,out}$.

Let us first consider universal termination. Assume that CW terminates universally, ans select a symbolic run $\rho^S = C_0^S \dots C_n^S$. We can then compute a chain of weakest preconditions $\psi_n, \psi_{n-1}, \dots \psi_0$ that have to be enforced to execute successfully CW and terminate in node n. In particular, $\psi_n ::= true$. Similarly, one can compute at each step, a weakest precondition $\psi_i^{in,out}$ needed at step i so that $\psi^{in,out}$ holds. Intuitively, $\psi_i^{in,out}$ describes the constraints between the initial dataset and the output dataset "consumed" at stage $i+1$ in ρ^S. If at one stage, $\psi_i \wedge \psi_i^{in,out}$ is not satisfiable, then ρ^S is not the signature of an actual run of CW that terminates properly, and we have found a witness of non-proper termination. We have assumed that $\psi^{in,out}$ was specified in a decidable fragment of FO. As computing the weakest precondition of a property in the existential, universal, BSR, SF fragment of FO gives a property in the same fragment, all ψ_i's and $\psi_i^{in,out}$'s are in a decidable fragment of FO. Then, the complexity will depend on the considered fragment, and on the fragment of FO used to specify inputs. As for universal termination, if inputs and $\psi^{in,out}$ are specified with the universal fragment of FO, then universal proper termination is in $co-2EXPTIME$, and in $co-3EXPTIME$ for the existential fragment (as one may alternate $\exists$ statements on outputs with $\forall$ statements inherited from obligation to prove non-emptiness of a dataset. Similar remark and complexity holds for the BSR fragment (separation of variables maintains an NEXPTIME complexity [37]). If $\psi^{in,out}$ is in SF, then checking proper universal termination is $co-K-fold-$exponential time, where $K = r^{K\tau_{cx}}$.

The proof and complexities for existential termination follow the same lines, yielding $2EXPTIME$ complexity when $\psi^{in,out}$ is written with the existential, fragment of FO, $3-EXPTIME$ complexity for when $\psi^{in,out}$ is written in the

universal or BSR fragments (as checking satisfiability for a BSR formula is in NEXPTIME [28]) and $K - fold-$exponential for SF formulas [37]. □