

# Data Centric Workflows for Complex Crowdsourcing Applications

Loïc Hérouët, Rituraj Singh, Zoltán Miklós

► **To cite this version:**

Loïc Hérouët, Rituraj Singh, Zoltán Miklós. Data Centric Workflows for Complex Crowdsourcing Applications. 2019. <hal-01976280>

**HAL Id: hal-01976280**

**<https://hal.inria.fr/hal-01976280>**

Submitted on 9 Jan 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Data Centric Workflows for Complex Crowdsourcing Applications

Hélouët, Loïc  
INRIA Rennes  
loic.helouet@inria.fr

Singh, Rituraj  
Univ. Rennes  
rituraj.singh@irisa.fr

Miklos, Zoltan  
Univ. Rennes  
zoltan.miklos@irisa.fr

## ABSTRACT

Crowdsourcing is a major paradigm to accomplish work that requires human skills, by paying a small sum of money and alluring workers whole across the globe. However, the targeted tasks at crowdsourcing platforms are relatively simple and independent work units. This work proposes a data centric workflow model for the design of complex crowdsourcing tasks. The model is called *complex workflows* and allows orchestration of simple tasks and concurrency. It handles data and crowdworkers and provides high-level constructs to decompose complex tasks into orchestrations of simpler subtasks. We first define the syntax and semantics of the model, and then consider its formal properties, starting with termination questions. We show that existential termination (existence of at least one terminating run) is undecidable. On the other hand, universal termination (whether *all* runs of a complex workflow terminate) is decidable. We then address correctness problems. We use FO formulas to specify dependencies imposed by a client between inputs and outputs of a workflow. If dependencies are specified with the separated fragment of FO, then universal correctness (whether all terminating runs satisfy dependencies) is decidable, and existential correctness (whether some terminating runs satisfy dependencies) is decidable under some semantic restrictions.

## CCS CONCEPTS

• Information systems → Crowdsourcing.

## KEYWORDS

Crowdsourcing; Data-centric workflows

### ACM Reference Format:

Hélouët, Loïc, Singh, Rituraj, and Miklos, Zoltan. 2019. Data Centric Workflows for Complex Crowdsourcing Applications. In *Proceedings of ACM SIGMOD/PODS International Conference on Management of Data (PODS'19)*. ACM, New York, NY, USA, Article 4, 22 pages. <https://doi.org/10.475/123.4>

## 1 INTRODUCTION

Crowdsourcing is a powerful tool to leverage intelligence of crowd to realize tasks where human skills still outperform

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).  
*PODS'19, July 2019, Amsterdam, The Netherlands*  
© 2019 Copyright held by the owner/author(s).  
ACM ISBN 123-4567-24-567/08/06.  
<https://doi.org/10.475/123.4>

machines [11]. It has been successful in contributive science initiatives, such as CRUK's Trailblazer<sup>1</sup>, Galaxy Zoo<sup>2</sup>, etc. Most often, a crowdsourcing project consists in deploying huge amount of work into tasks that can be handled by humans in a reasonable amount of time. Generally, tasks have the form of micro-tasks, which usually take a few minutes to an hour to complete. It can be labeling of images, writing scientific blogs, etc. The requester publishes the task on the platform with a small incentive (a few cents, reputation gain, goodies, etc.), and waits for the voluntary participation or bidding from the crowd. The micro-tasks proposed on crowdsourcing platforms are hence relatively simple, independent, cheap and repetitive. The next stage of crowdsourcing is to design more involved processes still relying on the vast wisdom of the crowd. Crowdsourcing markets such as Amazon Mechanical Turk<sup>3</sup> (AMT), Foule Factory<sup>4</sup>, CrowdFlower<sup>5</sup>, etc. already propose interfaces to access crowds, but the formal design and specification of crowd based complex processes is still in its infancy. Many projects cannot be described as collections of repetitive independent micro-tasks: they require specific skills and collaboration among participants. They shall hence be considered as complex tasks involving a workflow within a collaborative environment. The typical shape of such complex tasks is an orchestration of high-level phases (tag a database, then find relevant records, and finally write a synthesis). Each of these phases requires specific skills, can be seen at its level as a new objective on its own, and can be decomposed into finer choreographies, up to the level of assembly of micro-tasks. Within this setting, the challenges are the following: first, there is a discrepancy between the "project level", where clients of a crowd platform may have a good understanding of how to decompose their high-level projects into phases, and the micro-task level, where the power of existing crowdsourcing solutions can be used without knowing the high-level objectives of the projects. Transforming high-level phases into orchestrations of micro-tasks is also a difficult process. A second challenge is to exploit contributors skills and data collected along the orchestration, to improve the expressive power and accuracy of complex tasks. One possibility to implement high-level phases proposed by a client is to refine specifications to obtain a static orchestration of easy micro-tasks before execution of this low-level workflow. However, this solution lacks reactivity, and may miss some interesting skills of stakeholders who cannot realize directly

<sup>1</sup><https://www.cancerresearchuk.org>

<sup>2</sup><http://zoo1.galaxyzoo.org>

<sup>3</sup>[www.mturk.com](http://www.mturk.com)

<sup>4</sup><https://www.foulefactory.com>

<sup>5</sup><https://www.crowdfLOWER.com>

a particular task, but know how to obtain the result, or can bring data to enhance the information owned by the system. One can imagine for instance that collected data is used in real time to choose an orchestration and even the way tasks are decomposed. This calls for the integration of *higher-order schemes* in the realization of complex tasks. Last, clients may want guarantees on duration of their projects and on the returned results. It is hence interesting to consider termination and output correctness questions for complex tasks.

In this paper, we focus on orchestration of complex tasks in crowdsourcing environment, with higher order constructs allowing online decomposition of the tasks by crowdworkers. A complex task is defined as a workflow. At the very beginning, a coarse description is provided by the process requester, possibly with input data and with requirements on the expected output. Tasks in a workflow receive input data, and output data once realized. At each step, crowdworkers can decide to realize a task with the provided inputs, or decompose the task and its inputs into orchestrations of smaller work units. We first propose a model for complex tasks, allowing for the definition of data-centric workflows with higher-order schemes to refine tasks at runtime, and for the definition of constraints on inputs and outputs of the workflow. We then consider the question of termination: given a workflow, input data, a set of crowdworkers, allowed to transform input data or decompose tasks, is the workflow executable (or always executed) up to its end? We show that due to higher-order, complex workflows are Turing complete, and hence existence of a terminating run is not decidable. However, termination of all runs is decidable, and upon some sensible restrictions that forbid decomposition of the same type of task an arbitrary number of times, existential termination becomes decidable. As a third contribution, we consider *proper termination*, i.e., whether a complex workflow terminates and returns data that comply with client's requirements.

**Related Work :** Realization of complex tasks on crowdsourcing platforms is still a recent topic, but some works propose solutions for data acquisition and management or deployment of workload, mainly at the level of micro-tasks [7, 13]. Crowdforge uses Map-Reduce techniques along with a graphical interface to solve complex tasks [9]. Turkit [14] is a crash and rerun programming model. It built on an imperative language, that allows for repeated calls to services provided by a crowdsourcing platform. A drawback of this approach is that clients may not have the programming skills needed to design complex orchestrations of platform services. Turkomatic [12] is a tool that recruits crowd workers to help clients planning and solving complex jobs. It implements a Price, Divide and Solve (PDS) loop, that asks crowdworkers to divide a task into orchestrations of subtasks, and repeats this operation up to the level of micro-tasks. A PDS scheme is also used by [20] in a model based on hierarchical state machines. States represent complex tasks that can be divided into orchestrations of sub-tasks. Both approaches require monitoring of workflows by the client, which is cumbersome and does not match with the goal of providing a high-level service. These PDS oriented solutions have been validated empirically on case studies, but

formal analysis of tasks realization is not the main concern of these works.

Several formal models and associated verification techniques have been proposed in the past for data-centric systems or orchestration of tasks. Workflow nets [19] is a variant of Petri nets dedicated to business processes. They allow parallel or sequential execution of tasks, fork and join operations to create or merge a finite number of parallel threads. Tasks are represented by transitions. Workflow nets mainly deal with the control part of business processes, and data is not central for this model. Data-centric models and their correctness have also been considered. Guarded Active XML [1] (GAXML for short) is a specification paradigm where services are introduced in structured data. The model is defined as structured data that embed references to service calls. Services can modify data when their guard is satisfied, and replace a part of the data by some computed value that may also contain references to service calls. Though GAXML does not really address crowdsourcing nor tasks refinement, if services are seen as tasks, the replacement mechanism performed during calls can be seen as a form of task refinement. This model is very expressive, but restrictions on recursion allows for verification of Tree LTL (a variant of LTL where propositions are replaced by statements on the structured data). More recently, [2] 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 [17], and verification mechanisms for LTL-FO were proposed in [4, 10] for subclasses of artifacts with data dependencies and arithmetic. LTL-FO formulas are of the form  $\forall x_1, \dots, x_k, \phi$  where  $\phi$  is an LTL formula including FO statements. Variables are always universally quantified. [6] consider verification of LTL-FO for systems composed of peers that communicate asynchronously over possibly lossy channels and can modify (append/remove records from local databases). Unsurprisingly, queues makes LTL-FO undecidable, but bounding the queues allows for verification. The way data is handled in business artifacts is close to our model, and as for complex workflows, allows for data inputs during the lifetime of an artifact. However, artifacts mainly consider static orchestrations of guarded tasks, described as legal relations on datasets before and after execution of a task, and does not consider higher-order constructs such as runtime task refinement. Further, LTL-FO verification focuses mainly on dynamics of systems (termination, reachability) but does not address correctness.

This paper is organized as follows: Section 2 introduces our model. Section 3 defines its operational semantics, and section 4 addresses the termination question. Section 5 considers proper termination of complex workflows, before conclusion. Due to lack of space, some proofs are only sketched, and are provided in appendix.

## 2 COMPLEX WORKFLOWS

In this section, we formalize the notion of complex workflow, and give its semantics through operational rules. This model

is inspired by artifacts systems [4], but uses higher-order constructs (task decomposition), and deals with human resources within the system (the so-called *crowdworkers*). The context of use of the complex workflow is the following : 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.

We furthermore assume that this client can reward contribution of human stakeholders up to a certain budget, that he can input data to the system, and that he may have a priori knowledge on the relation between the contents of his input and the plausible outputs returned after completion of his complex task. In its simplest form, this type of application can be an elementary tagging task for a huge database. This type of application was met in citizen science initiatives such as Galaxy zoo<sup>6</sup>, but several types of applications such as opinion polls, citizen participation, etc. can be seen as complex crowdsourcing tasks.

We start with an 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. He decomposes this ranking into three 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: the output is an instance of a relational schema  $R = (name, cites, category, score)$ , 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, for an output to be consistent, every actor appearing in the final database should have a score and a number of *cites* greater than 0. From this example, one can notice that there are several ways to collect actors names, several ways to associate a category tag, to vote, etc. but that the clients needs are defined in terms of high-level tasks, without information on how the crowd will be used to fulfill the demand.

## 2.1 Workflow Entities

A *complex workflow* is defined as an orchestration of *tasks*, specified by a *client* to process *input* data and return an *output* dataset. Tasks that can be accomplished by either humans, i.e. *workers* if they require human skills or be automated tasks that can be executed by machines. Additionally, worker's task can be an elementary or a complex task.

We assume a fixed and finite pool  $U$  of workers, and an a priori finite list of competences **comp**. Each worker  $u \in U$  can complete or refine some tasks according to its *skills*  $sk(u) \subseteq \mathbf{comp}$ . During the execution of a complex workflow, we will consider that each worker is engaged in the execution of at most one task. 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 of the crowdsourcing platform, a very basic atomic task that can be easily accomplished by a single worker (tagging images, for instance), a task that can be fully automated, or a complex task that still requires a small orchestration of subtasks 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 task* which need to be decomposed into an orchestration of smaller subtasks to produce an output, and  $\mathcal{T}_{aut}$  is a set of automated tasks that are performed by a machine (for instance some database operation (selection, union, projection, etc.) executed as an SQL query). Tasks in  $\mathcal{T}_{aut}$  do not require contribution of a worker to produce output data from input data, and tasks in  $\mathcal{T}_{ac}$  and  $\mathcal{T}_{aut}$  cannot be refined. We impose constraints on skills required to execute a task with a map  $T_{cs} : \mathcal{T} \rightarrow 2^{\mathbf{comp}}$ , depicting the fact that a worker  $u$  is allowed to execute or refine task  $t$  if  $T_{cs}(t) \cap sk(u) \neq \emptyset$ . For decomposition of a task  $t \in \mathcal{T}_{cx}$ , each worker  $u$  with appropriate skills knows how to refine  $t$ , and possesses several orchestrations depicting possible refinements of  $t$ , i.e. a set of finite workflows  $Profile(t, u)$ . Let us illustrate refinement with an example. Assume a task  $t \in \mathcal{T}_{cx}$  which role is to tag a (huge) dataset  $D_{in}$ . Then,  $Profile(t, u)$  contains a workflow that first decomposes  $D_{in}$  into  $K$  small tables, then inputs these tables to  $K$  tagging tasks in  $\mathcal{T}_{ac}$  that can be performed by humans, and finally aggregates the  $K$  obtained results. Similarly, a refinement in  $Profile(t, u)$  can simply replace  $t$  by a single atomic tagging task  $t' \in \mathcal{T}_{ac}$ , meaning that  $u$  wants the task to be performed by a single worker.

In addition to the notion presented above, crowdsourcing platforms often consider incentives, i.e. the benefit provided to the worker for performing a particular task. Incentive mechanism can be intrinsic (Self motivation, Gamification, Share Purpose, Social cause, etc.) as well as extrinsic (Tailor rewards, Bonus, Promote Workers, etc.) [5]. In this paper, we leave this notion of incentives apart, and consider that all users are equally eager to perform all tasks that are compatible with their competences. One shall however keep in mind that setting incentives appropriately is a key issue to complete successfully a workflow: associating high rewards to important or blocking tasks is a way to maximize the probability that these tasks will be realized by a worker.

## 2.2 Workflows

*Definition 2.1 (Workflow).* A *workflow* is a labeled acyclic graph  $W = (N, \rightarrow, \lambda)$  where  $N$  is a set of nodes, representing 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 of  $W$ . A node of  $W$  is a *source* iff it has no predecessor, and a *sink* iff it has no successor. We require that a workflow has at most one sink node, denoted  $n_f$ .

In the rest of the paper, we will consider that  $U$  and  $\mathcal{T}$  are fixed, and we will denote by  $\mathcal{W}$  the set of all possible workflows. Intuitively, if  $(n_1, n_2) \in \rightarrow$ , then a task of type  $\lambda(n_1)$  represented by  $n_1$  must be completed before a task

<sup>6</sup><http://zoo1.galaxyzoo.org/>

of type  $\lambda(n_2)$  represented by  $n_2$ , and that data computed by  $n_1$  is used as input for  $n_2$ . We denote  $\text{min}(W)$  the set of sources of  $W$ , by  $\text{succ}(n)$  the set of successors of a node  $n$ , and by  $\text{pred}(n)$  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 dataset to its successors. With this convention, one can model situations where a large database is to be split into smaller datasets of reasonable sizes and sent to tagging tasks that needs to be completed by workers. We denote by  $W \setminus \{n\}$  the restriction of  $W$  to  $N \setminus \{n\}$ , that is, a workflow from which we remove node  $n$  and all edges which origins or goals are node  $n$ . 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  $n$  of  $W$  to the sink  $n_f$ . This property prevents launching tasks which results are never used to build an answer to a client.
- for every workflow  $W = (N, \rightarrow, \lambda) \in \text{Profile}(t, u)$ , the labeling  $\lambda$  is injective. This results in no loss of generality, as one can create copies of a task for each node in  $W$ , but simplifies proofs and notations afterwards. Further,  $W$  has a unique source node  $\text{src}(W)$ .

*Definition 2.2 (Refinement).* Let  $W = (N, \rightarrow, \lambda)$  be a workflow,  $W' = (N', \rightarrow', \lambda')$  be a workflow with a unique source node  $n'_{\text{src}} = \text{src}(W')$  and a unique sink node  $n'_f$  and such that  $N \cap N' = \emptyset$ . The replacement of  $n \in N$  by  $W'$  in  $W$  is the workflow  $W_{[n/W']} = (N_{[n/W]}, \rightarrow_{[n/W]}, \lambda_{[n/W]})$ , where:

- $N_{[n/W']} = (N \setminus \{n\}) \cup N'$
- $\rightarrow_{[n/W]} = \rightarrow' \cup \{(n_1, n_2) \in \rightarrow \mid n_1 \neq n \wedge n_2 \neq n\} \cup \{(n_1, n'_{\text{src}}) \mid (n_1, n) \in \rightarrow\} \cup \{(n'_f, n_2) \mid (n, n_2) \in \rightarrow\}$
- $\lambda_{[n/W]}(n) = \lambda(n)$  if  $n \in N$ ,  $\lambda'(n)$  otherwise

To illustrate the notion of refinement, consider the example of Figure 1. A workflow  $W_{t_2}$  is used to refine task  $t_2$  in the workflow appearing in box  $C_1$ .

### 2.3 Data

The data used in complex workflow refer to data provided as input to the system by a client, to the data conveyed among successive tasks, and to data returned after completion of a workflow, that is returned to the client. Data is organized in tables or datasets, that follow some *relational schemas*. We assume finite set of domains  $\mathbf{dom} = \text{dom}_1, \dots, \text{dom}_s$ , a finite set of attribute names  $\mathbf{att}$  and a finite set of relation names  $\mathbf{relnames}$ . Each attribute  $a \in \mathbf{att}$  is associated with a domain  $\text{dom}(a) \in \mathbf{dom}$ . A *relational schema* (or table) is a pair  $rs = (rn, A)$ , where  $rn$  is a relation name and  $A$  denotes a finite set of attributes. Intuitively, attributes are column names in a table. The arity of  $rs$  is the size of its attributes set. An *record* of a relational schema  $rs = (rn, A)$

is tuple  $rn(v_1, \dots, v_{|A|})$  where  $v_i \in \text{dom}(a_i)$  (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.

Execution of a task  $t = \lambda(n)$  in a workflow builds on input data to produce output data. The data input to node  $n$  with  $k$  predecessors is a list of datasets  $\mathcal{D}^{in} = D_1^{in}, \dots, D_k^{in}$ . For simplicity, we consider that predecessors (resp. successors of a node) are ordered, and that dataset  $D_i^{in}$  input to a node is the data produced by predecessor  $n_i$ . Similarly, for a node with  $q$  successors, the output produced by a task will be  $\mathcal{D}^{out} = D_1^{out} \dots D_q^{out}$ . As for inputs, we will consider that dataset  $D_i^{out}$  is the data sent to the  $i^{\text{th}}$  successor of node  $n$ . The way output data is produced by task  $t = \lambda(n)$  and propagated to successor nodes depends on the nature of the task. If  $t$  is an automated task, the outputs are defined as a deterministic function of inputs, i.e.,  $\mathcal{D}^{out} = f_t(\mathcal{D}^{in})$  for some deterministic function  $f_t$ . We will allow automated tasks executions only for nodes which inputs are not empty. In the rest of the paper, we will consider that automated tasks perform simple SQL operation : projections on a subset of attributes, selection of records that satisfy some predicate, record insertion or deletion.

To simplify workflows refinements, we will consider particular *split nodes*, i.e. have a single predecessor, a fixed number  $k$  of successors, and are attached a task  $t \in \mathcal{T}_{aut}$  that transforms a **non-empty** input  $\mathcal{D}^{in}$  into a list  $\mathcal{D}^{out} = D_1^{out} \dots D_k^{out}$ . Note that  $\mathcal{D}^{out}$  needs not be a partition of  $\mathcal{D}^{in}$  not to define distinct output datasets. To refer to the way each  $D_i^{out}$  is computed, we will denote by  $\text{spl}_t^{(i)}$ , the function that associates to  $\mathcal{D}^{in}$  the  $i^{\text{th}}$  output produced by the splitting task (i.e.  $\text{spl}_t^{(i)}(\mathcal{D}^{in}) = D_i^{out}$ ). Consistently with the non-empty inputs requirement the input dataset to split cannot be empty to execute such splitting task. Similarly, we will consider *join nodes*, which role is to automatically aggregate multiple inputs from  $\mathcal{D}^{in}$ . Such aggregation nodes can simply perform union of datasets with the same relational schema, or a more complex join. Consider a node  $n$  with several predecessors  $n_1, \dots, n_k$ , and a single successor  $s$ . Let  $D_{in} = D_1.D_2 \dots D_k$ , where all  $D_i$ 's have the same relational schema. Then we can define a join node by setting  $f_t(\mathcal{D}^{in}) = \bigcup_{i \in \{1..|D^{in}|\}}$ . Consistently with the non-empty inputs requirement, none of the the input datasets is empty when a join is performed.

For an atomic task  $t \in \mathcal{T}_{ac}$  attached to a node  $n$  of a workflow and executed by a particular user  $u$ , data  $D_{in}$  comes from preceding nodes, but the output depends on the user. Hence, execution of task  $t$  by user  $u$  produces an output  $\mathcal{D}^{out}$  chosen non-deterministically from a set of possible outputs  $F_{t,u}(D_{in})$ . For the rest of the paper, we will assume that the legal contents of  $F_{t,u}(D_{in})$  is defined as a first order formula  $\phi_{t,in,out}$  that holds for datasets  $\mathcal{D}^{in} = D_1^{in} \dots D_k^{in}$  and  $\mathcal{D}^{out} = D_1^{out} \dots D_k^{out}$  if  $\mathcal{D}^{out} \in F_{t,u}(D_{in})$ . This way to depict legal productions of an user  $u$  to make non-deterministic choices, to insert new records in a dataset...

*Definition 2.3.* A *Complex Workflow* is a tuple  $CW = (W_0, \mathcal{T}, T_{cs}, \mathcal{U}, sk, \mathcal{P})$  where  $\mathcal{T}$  is a set of tasks,  $\mathcal{U}$  a finite set of workers,  $\mathcal{P} : \mathcal{T} \times \mathcal{U} \rightarrow 2^{\mathcal{W}}$  associates to pairs  $(t, u)$  of complex tasks and workers a set  $Profiles(t, u) = \mathcal{P}(t, u)$  of possible workflows that  $u$  can use to refine  $t$ ,  $sk$  defines workers competences, and  $T_{cs}$  gives the competences needed to refine a task.  $W_0$  is an initial workflow, that contains a single source node  $n_i$  and a single sink node  $n_f$ .

### 3 OPERATIONAL SEMANTICS

The execution of a complex workflow consists in realizing all its tasks, following the order given by the dependency relation  $\rightarrow$  in the orchestration. 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  $n_f$ .

An *assignment* for a workflow  $W = (N, \rightarrow, \lambda)$  is a partial map  $Ass : N \rightarrow \mathcal{U}$  such that for every node  $n \in Dom(Ass)$ ,  $T_{cs}(\lambda(n)) \cap sk(Ass(n)) \neq \emptyset$  (worker  $Ass(n)$  has competences to complete task  $\lambda(n)$ ). We furthermore require map  $Ass$  to be injective, i.e. a worker is involved in at most one task. We say that  $u \in \mathcal{U}$  is free if  $u \notin Ass(N)$ . If  $Ass(n)$  is not defined, and  $u$  is a free worker,  $Ass \cup \{(n, u)\}$  is the map that assigns node  $n$  to worker  $u$ , and remains unchanged for every other node. Similarly,  $Ass \setminus \{n\}$  is the restriction of  $Ass$  to  $N \setminus \{n\}$ .

A *data assignment* for  $W$  is a function  $\mathbf{Dass} : N \rightarrow (DB \uplus \{\emptyset\})^*$ , that assigns a sequence of input datasets to nodes in  $W$ . 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. We denote by  $\mathbf{Dass}(n)_{[i/X]}$  the sequence obtained by replacement of  $D_i$  by  $X$  in  $\mathbf{Dass}(n)$ .

*Definition 3.1 (Configuration).* A *configuration* of a complex workflow is a triple  $C = (W, Ass, \mathbf{Dass})$  where  $W$  is a workflow depicting remaining tasks that have to be completed,  $Ass$  is an assignment, and  $\mathbf{Dass}$  is a data assignment.

A complex workflow execution starts from the *initial configuration*  $C_0 = (W_0, Ass_0, \mathbf{Dass}_0)$ , where  $Ass_0$  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  $C_f = (W_f, Ass_f, \mathbf{Dass}_f)$  such that  $W_f$  contains only node  $n_f$ ,  $Ass_f$  is the empty map, and  $\mathbf{Dass}_f(n_f)$  represents the dataset that has to be returned to the client, and that has been assembled during the execution of all nodes preceding  $n_f$ . The intuitive understanding of this type of configuration is that  $n_f$  needs not be executed, and simply terminates the workflow by returning 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 the following 4 rules. Rule 1 defines the task assignment 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.

**Rule 1 (WORKER ASSIGNMENT):** A worker  $u \in \mathcal{U}$  is assigned a task  $t = \lambda(n)$  if  $t \notin T_{aut}$ . 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 an user even if it does not have input data yet, and is not yet executable.

$$\frac{n \notin Dom(Ass) \wedge u \notin coDom(Ass) \wedge sk(u_j) \cap T_{cs}(\lambda(n)) \neq \emptyset \wedge \lambda(b) \notin T_{aut}}{(W, Ass, \mathbf{Dass}) \rightarrow (W, Ass \cup \{(n, u)\}, \mathbf{Dass})} \quad (1)$$

**Rule 2 (ATOMIC TASK COMPLETION):** An atomic task  $t = \lambda(n)$  can be executed if node  $n$  is minimal in the workflow, it is assigned to a worker  $u = Ass(n)$  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 free.

$$\frac{\begin{aligned} &n \in min(W) \wedge \lambda(n) \in T_{ac} \wedge Ass(n) = u \\ &\wedge \mathbf{Dass}(n) \notin DB^*.\emptyset.DB^* \\ &\wedge \exists \mathcal{D}^{out} = D_1^{out} \dots D_k^{out} \in F_{\lambda(n), u}(\mathbf{Dass}(n)), \\ &\mathbf{Dass}' = \mathbf{Dass} \setminus \{(n, \mathbf{Dass}(n))\} \cup \\ &\quad \{(n_k, \mathbf{Dass}(n_k)_{[j/D_k^{out}]}) \mid n_k \in succ(n)\} \\ &\wedge n \text{ is the } j^{th} \text{ predecessor of } n_k \end{aligned}}{(W, Ass, \mathbf{Dass}) \xrightarrow{\lambda(n)} (W \setminus n, Ass \setminus \{(n, u)\}, \mathbf{Dass}')} \quad (2)$$

**Rule 3 (AUTOMATIC TASK COMPLETION):** An automatic 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 an user, and that the produced outputs are a deterministic function of task inputs.

$$\frac{\begin{aligned} &n \in min(W) \wedge \lambda(n) \in T_{aut} \wedge \mathbf{Dass}(n) \notin DB^*.\emptyset.DB^* \\ &\wedge \mathcal{D}^{out} = f_{\lambda(n), u}(\mathbf{Dass}(n)) = D_1^{out} \dots D_k^{out}, \\ &\mathbf{Dass}' = \mathbf{Dass} \setminus \{(n, \mathbf{Dass}(n))\} \cup \\ &\quad \{(n_k, \mathbf{Dass}(n_k)_{[j/D_k^{out}]}) \mid n_k \in succ(n)\} \\ &\wedge n \text{ is the } j^{th} \text{ predecessor of } n_k \end{aligned}}{(W, Ass, \mathbf{Dass}) \xrightarrow{\lambda(n)} (W \setminus n, Ass, \mathbf{Dass}')} \quad (3)$$

**Rule 4 (COMPLEX TASK REFINEMENT):** The refinement of a node  $n$  with  $t = \lambda(n) \in T_{cx}$  by worker  $u = Ass(n)$  replaces node  $n$  by a workflow  $W_s = (N_s, \rightarrow_s, \lambda_s) \in Profile(t, u)$ . 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.

$$\frac{\begin{aligned} &t = \lambda(n) \in T_{cx} \wedge W_s \in Profile(t, Ass(n)) \\ &\wedge \mathbf{Dass}'(min(W_s)) = \mathbf{Dass}(n) \\ &\wedge \forall x \in N_s \setminus min(W_s), \mathbf{Dass}'(x) = \emptyset^{Pred(x)} \\ &\wedge Ass' = Ass \setminus \{(n, Ass(n))\} \end{aligned}}{(W, Ass, \mathbf{Dass}) \xrightarrow{ref(n)} (W_{[n/W_s]}, Ass', \mathbf{Dass}')} \quad (4)$$

In our framework, a worker can refine a task if she thinks the task is too complex to be handled by a single person. However, the definition of a *complex* task is very subjective and varies from one worker to another. Note also that refinement is not mandatory: a worker can replace a node  $n$  with another node labeled by task  $t \in \mathcal{T}_{cx}$  by a node labeled by an equivalent task  $t' \in \mathcal{T}_{ac} \cup \mathcal{T}_{aut}$ .

Figure 1 gives an example of rules application. Workflow nodes are represented by circles, tagged with a task name representing map  $\lambda$ . The dependencies are represented by plain arrows between nodes. User assignments are represented by dashed arrows from an user name  $u_i$  to its assigned task. Data assignment are represented by double arrows from a dataset to a node. The top-left part of the figure is a configuration  $C_0 = (W_0, Ass_0, \mathbf{Dass}_0)$  composed of an initial workflow  $W_0$ , an empty map  $Ass_0$  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  $C_1 = (W_1, Ass_1, \mathbf{Dass}_1)$  obtained by assigning user  $u_1$  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 user  $u_1$  decides to refine task  $t_2$  according to the profile  $W_{t_2,1} \in Profile(t_2, u_1)$  (Rule 4). Workflow  $W_{t_2,1}$  is the part of the Figure contained in the Grey square.

We will 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 3.2 (Run).* A run  $\rho = C_0.C_1 \dots C_k$  of complex workflow  $W$  with users  $\mathcal{U}$  is a finite sequence of configurations such that  $C_0$  is an initial configuration, and for every  $i \in 1 \dots k$ , there exists a move from  $C_{i-1}$  to  $C_i$ . A run is *maximal* if  $C_k$  has no successor. A maximal run is *terminated* iff  $C_k$  is a final configuration, and it is *deadlocked* otherwise.

In the rest of the paper, we denote by  $\mathcal{Runs}(CW, D_{in})$  the set of maximal runs originating from *initial configuration*  $C_0 = (W_0, Ass_0, \mathbf{Dass}_0)$  (where  $\mathbf{Dass}_0$  associates dataset  $D_{in}$  to node  $n_i$ ). We denote by  $\mathcal{Reach}(CW, D_{in})$  the set of configurations that can be reached from  $C_0$ . Along a run, the datasets in use can grow, and the size of the workflow can also increase, due to decomposition of tasks. Hence,  $\mathcal{Reach}(CW, D_{in})$  and  $\mathcal{Runs}(CW, D_{in})$  need not be finite. Even when  $\mathcal{Reach}(CW, D_{in})$  is finite, a complex workflow may exhibit infinite cyclic behaviors.

Moves in executions of complex workflows consists in workflow rewriting, computation of datasets, and appropriate transfer of datasets from one task to another. Complex tasks and their refinement can encode recursive unbounded recursive schemes. For instance, consider a simple linear workflow composed of three nodes :  $n_i, n_f, n_1$  where  $n_1$  is attached task  $\lambda(n_1) = t_1$ , and such that  $\rightarrow = \{(n_i, n_1), (n_1, n_f)\}$ . Let us assume that our system has a single user, and that this user has a decomposition profile  $(t_1, W_{t_1})$  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  $\rightarrow = \{(w_2, w_1), (w_1, w_f)\}$ . Then, after application of rule  $R_1$  (assigning user 1 to the node that

carries task  $t_1$  and  $R_4$  (replacing  $t_1$  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, but one can easily find unbounded recursive schemes with unboundedly growing of data (for instance if  $\lambda(w_f)$  adds a record to some dataset). Hence, without restriction, complex workflows define transitions systems of arbitrary size, with growing data or workflow components, and with cycles.

## 4 TERMINATION

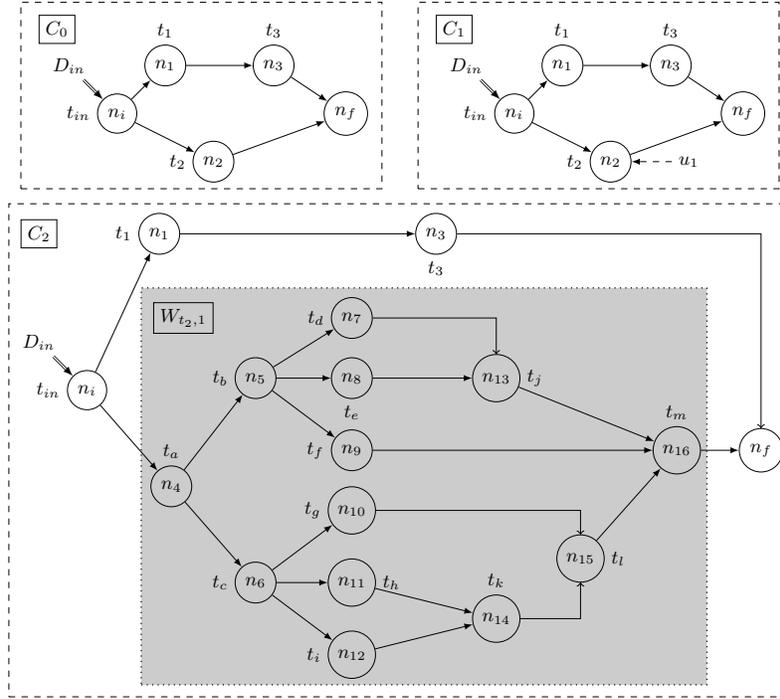
Complex workflows use the knowledge and skills of crowd users to complete a task starting from input data provided by a client. Now, a workflow may never reach a final configuration. This can be due to particular data input by workers that cannot be processed properly by the workflow, to infinite recursive schemes appearing during the execution, to deadlocked situations due to missing worker competences... It is hence important to detect whether some/all runs of a system eventually reach a *final configuration* in  $\mathcal{C}_f$ .

*Definition 4.1 (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  $\mathcal{Runs}(CW, D_{in})$  that is terminated.  $CW$  *terminates universally* on input  $D_{in}$  iff all runs in  $\mathcal{Runs}(CW, D_{in})$  are terminated. Similarly,  $CW$  *terminates (universally or existentially)* on input set  $\mathcal{D}_{in}$  iff  $CW$  *terminates* on every input  $D_{in} \in \mathcal{D}_{in}$ .

We describe sets of inputs  $\mathcal{D}_{in}$  symbolically with a decidable fragment of FO (e.g. the separated fragment introduced later in this section). Given a complex workflow  $CW$  the existential termination problem consists in checking whether some run of  $CW$  terminates for an input  $D_{in}$  (or all inputs in set  $\mathcal{D}_{in}$ ). The universal termination problem consists in checking whether all runs of  $CW$  terminate for an input  $D_{in}$  (or all inputs in set  $\mathcal{D}_{in}$ ). Solving these problems is a way to ensure that an answer to a client (an output dataset  $D_{out}$ ) can be returned, or will always be returned. Existential and universal termination are of different natures. The former is undecidable while the latter is decidable.

**THEOREM 4.2.** *Existential termination of complex workflows is an undecidable problem.*

**SKETCH.** Complex workflows can simulate any two counters machine. The encoding proceeds as follows: each instruction  $i$  of the counter machine is encoded as a specific task  $t_i$ , that can be refined by only one user  $u_i$ . The workflow  $W_i$  chosen for refinement by  $u_i$  is then executed until it contains a single node representing the next instruction. Counters are encoded as the number of occurrences of specific tags  $c_1, c_2$  in a field of a dataset. When simulating a zero test and decrement instruction  $i$ , user  $u_i$  has to guess whether the value of a counter is zero or not (this is encoded as a choice



**Figure 1: Complex workflow execution.**  $C_0$  represents the initial configuration with data  $D_{in}$  allocated to node  $n_i$ .  $C_1$  is the successor of  $C_0$ : user  $u_1$  is allocated to node  $n_2$ , and  $t_2 = \lambda(n_2)$  is a complex task.  $C_2$  depicts the configuration after refinement of node  $n_2$  by a new workflow  $W_{t_2}$  (shown in the Grey rectangle).

of a particular workflow to refine  $t_i$ ). If the user does the wrong guess, the execution deadlocks. Otherwise, the execution always proceeds to the next instruction. More details on this encoding are provided in Appendix A.1.

The question of termination for a set of initial datasets is also undecidable (it suffices to write  $D_{in} = \{D_{in}\}$  to get back to the former termination question).  $\square$

Universal termination is somehow an easier problem. We show in this section that it is indeed decidable. We proceed in several steps. We first define a *symbolic execution tree* representing possible sequences of moves starting from the initial configuration. This tree is a transition system that abstracts away the data part. Each path of the tree defines a *signature* for a set of runs with the same sequence of moves but different data assignments. It is a priori an unbounded structure, but we show that a complex workflow does not terminate universally if it allows unbounded recursion. If a complex workflow does not allow unbounded recursion, then its execution tree is finite, and if this execution tree contains a deadlocked path, then one has a witness for non-termination. A deadlocked path can be either path ending with remaining tasks, but that cannot be extended, or path that ends in a configuration from which a task split is needed, but where emptiness of a dataset can occur and prevent the split. We show that existence of such a run can be effectively decided from paths of the execution tree, and by computing the conditions needed to reach a configuration with empty input to a split node. More precisely, given a first order formula  $\phi$

depicting the contents of datasets, given a signature  $\sigma$  in the execution tree, one can check the existence of an actual run  $\rho$  with signature  $\sigma$  that ends in a configuration that satisfies  $\phi$ . The crux in the proof is the possibility to compute a sequence of weakest preconditions that have to be satisfied at each step of  $\rho$  to guarantee existence of an actual run. We show first that these preconditions can be effectively computed, and then that the fragment of FO that have to be used to verify termination belongs to the decidable separated fragment of FO, for which satisfiability is decidable.

Each configuration  $C_i = (W_i, Ass_i, \mathbf{Dass}_i)$  in a run  $\rho$  contains a workflow  $W_i$  with a finite number of nodes, assignments of workers to tasks, and data assignments. An execution  $\rho = C_0 \dots C_k$  of a complex workflow terminates iff the reached configuration is of the form  $C_k = (W_f, Ass_f, \mathbf{Dass}_f)$  where  $W_f$  contains only the final node of a workflow. Checking termination hence amounts to checking whether one can reach such a configuration.

A move from  $C_i$  to  $C_{i+1}$  leaves the number of nodes unchanged (application of user assignment rule  $R_1$ ), decreases the number of nodes ( execution of an atomic task (R2), or of an automated task (R3)), or refines a node in  $W_i$ . Only in this latter case, the number of nodes may increase. However, application of decomposition rule  $R_4$  to a node  $n$  with  $\lambda(n) = t_i$  can occur at most  $KD(i)$  times due to the restricted decomposition rule. Note also that without application of rule  $R_4$  that creates new nodes, the number of applications of each rule  $R_1, R_2, R_3$  is bounded by the size of the workflow.

The set of possible transformations of  $W_i$  and  $Ass_i$  occurring from  $C_i$  is bounded, and one can design a tree of finite degree depicting possible applications of semantics rules.

*Definition 4.3 (Symbolic Execution Tree).* The *Symbolic execution tree* (SET for short) of a complex workflow  $CW = (W_0, \mathcal{T}, T_{cs}, \mathcal{U}, sk, \mathcal{P})$  is a pair  $\mathcal{B} = (V, E)$ , where  $E \subseteq V \times V$  is a set of edges,  $V$  is a set of vertices of the form  $V_i = (W_i, Ass_i, \mathbf{Dass}^S)$ , where  $W_i = (N_i, \rightarrow_i, \lambda_i)$  and  $Ass_i : N_i \rightarrow \mathcal{U}$  are the usual workflow and user assignment relations, and  $\mathbf{Dass}^S$  associates a sequence of relational schemas to minimal nodes of  $W_i$ .

For every node  $n$  that is minimal in  $W_i$ , the meaning of  $\mathbf{Dass}^S(n) = rs_1 \dots rs_k$  is that task attached to node  $n$  takes as inputs datasets  $D_1 \dots D_k$  where each  $D_i$  conforms to relational schema  $rs_i$ . Notice that it is sufficient to know  $\lambda(n)$  to obtain  $\mathbf{Dass}^S(n)$ . Edges are built as follows: Let  $V_i = (W_i, Ass_i, \mathbf{Dass}^S)$  and  $V_j = (W_j, Ass_j, \mathbf{Dass}^S)$ . Then  $(V_i, V_j) \in E$  if one of the following situations holds:

- there exists  $u \in \mathcal{U}$  and  $n \in W_i$ ,  $Ass_i^{-1}(u) = \emptyset$ ,  $W_j = W_i$ ,  $\mathbf{Dass}^S = \mathbf{Dass}^S$  and  $Ass_j = Ass_i \uplus \{(n, u)\}$
- there exists  $n \in \min(W_i)$  with successors  $n_1, \dots, n_k$ ,  $W_j = W_i \setminus \{n\}$   $\mathbf{Dass}^S$  assigns to each successor  $n_j$  the relational schema corresponding to  $F_j(D_n)$
- there exists  $n \in \min(W_i)$   $\lambda(n)$  is a complex task, and  $Ass(n) = u$ ,  $W_j$  is the workflow obtained by replacement of  $n$  in  $W_i$  by a workflow in  $\mathcal{P}(\lambda(n), u)$ .  $\mathbf{Dass}^S$  assigns to each successor  $n_j$  the relational schema corresponding to  $F_j(D_n)$

One can consider paths in  $\mathcal{B}$  as "symbolic executions", i.e. executions where the contents of data is not made explicit. Given a run  $\rho = C_0 \dot{C}_1 \dots C_K$ , with  $C_i = (W_i, Ass_i, \mathbf{Dass}_i)$  we define the signature of  $\rho$  as the sequence of triples of the form  $(W_0, Ass_0, \mathbf{Dass}^S)$ , where  $\mathbf{Dass}^S$  is the map that assigns to minimal nodes in  $W_j$  the relational schema of their input datasets. Clearly,  $\mathcal{B}$  collects all signatures of all runs of a complex workflow. However, it is not necessarily finite.

*Definition 4.4 (Deadlocks, Potential deadlocks).* Let  $\mathcal{B} = (V, E)$  be a SET. A vertex of  $\mathcal{B}$  is *final* if its workflow part consists of a single node  $n_f$ . It is a *deadlock* if it has no successor. It is a *potential deadlock* iff a split action can occur from this node.

We use the term potential deadlock because distribution of data can be performed only when the input of a node that does this split is not empty. Let  $V_i = (W_i, Ass_i, \mathbf{Dass}_i)$  be a potential deadlock node. Let  $S = \{n_1, \dots, n_k\} \subseteq \min(W_i)$  be the set of minimal nodes that represent data splitting, i.e. that are assigned an user and have several successors in  $W_i$ . Let  $\Pi = V_0 \dots V_i$  be the path from the root of the tree to a potential deadlock node  $V_i$ . Even if vertex  $V_i$  has a successor  $V_k$ , obtained by executing a data distribution (i.e. executing a split task attached to some node  $n_j \in S$ ), it can be the case that  $\mathbf{Dass}_i$  assigns an empty input to  $n_j$  in an actual run of  $CW$  with signature  $\Pi$ . Hence, some of the splits depicted in  $\mathcal{B}$  may not be realizable. If executing task  $\lambda(n_j)$  the only

possible action from  $V_i$  and if a run with signature  $\Pi$  ends in a configuration where  $\mathbf{Dass}_i(n_j)$  is of the form  $D_1 \dots \emptyset \dots D_q$  then the run is deadlocked. However, if all runs with signature  $\Pi$  end with data assignments that affect non-empty sequences of datasets to all nodes of  $S$ , then  $V_i$  will never cause a real deadlock. Note also that when  $V_i$  is a potential deadlock, there exists necessarily a path  $V_i.V_{i+1} \dots V_{i+h}$  in  $\mathcal{B}$  where the only actions allowed from  $V_{i+h}$  are executions of splitting tasks. To detect that a potential deadlock can lead to a real deadlock, one has to answer the following question : is there an execution  $\rho = C_0 \dots C_i$  such that the execution of  $\rho$  has signature  $\Pi$  and such that  $\mathbf{Dass}_i(n_j)$  contains an empty dataset for some split node  $n_j \in \min(W_i)$  ?

PROPOSITION 1. *A complex workflow terminates universally iff the following conditions hold:*

- i) *Its symbolic execution tree is finite (there is no unbounded recursion)*
- ii) *there exists no path  $V_0 \dots V_i$  in the symbolic execution tree such that  $V_i$  is a deadlock*
- iii) *there exists no run with signature  $V_0 \dots V_i$  where  $V_i$  is a potential deadlock, with  $D_k = \emptyset$  for some  $D_k \in \mathbf{Dass}(n_j)$  and for some minimal split node  $n_j$  of  $W_i$ .*

Condition *i*) can be easily checked, by checking existence of reachable cycles in a graph that has tasks as vertices, and connects two tasks  $t, t'$  if  $t$  can be replaced by a workflow that contains  $t'$  (see Appendix A.3 for details). As soon as the symbolic execution tree is finite, checking condition *ii*) is a simple exploration: all leaves shall be vertices depicting families of final configurations.

Let us now show how to check the last condition *iii*). Let  $V_i$  be a vertex of  $\mathcal{B}$  and let  $n_j$  be a node that is attached a split task in  $W_i$ . We want to check if there exists an actual run  $\Pi = C_0 \dots C_i$  with signature  $V_0 \dots V_i$  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 an FO formula of the form  $\psi_i ::= \nexists x_1, \dots, x_{|A_j|}, rm_k(x_1, \dots, x_{|A_j|}) \in D_k$ . For  $\psi_i$  to hold in configuration  $C_i$ , inputs of minimal nodes in  $W_{i-1}$  may have to satisfy some constraint  $\psi_{i-1}$ . Depending on the nature of the move from  $V_{i-1}$  to  $V_i$ , the preconditions on inputs at step  $i-1$  differ. Let  $m_i$  denote the nature of move from  $V_{i-1}$  to  $V_i$ . We denote by  $wp[m_i]\psi_i$  the weakest precondition needed on inputs of minimal nodes in  $V_{i-1}$  (and hence also in  $C_{i-1}$ ) such that  $\psi_i$  holds on  $V_i$  (and hence also in  $C_i$ ). We can show (Proposition 2) that all needed constraints can be encoded as first order formulas, and that all preconditions, regardless of the type of move, can be effectively computed.

Now, to check that a path of  $\mathcal{B}$  with signature  $V_0 \dots V_i$  violates condition *iii*), we need to compute inductively all preconditions needed to reach a configuration where this split operation fails. We start from the assumption that some input  $D_k$  is empty when trying to execute the split in  $C_i$ , and compute backwards the conditions  $\psi_j = wp[m_{j+1}]\psi_{j+1}$  needed at each step  $j \in i-1 \dots 1$  to eventually reach a situation where  $D_k = \emptyset$  at step  $i$ . If any condition computed

this way (say at step  $j < i$ ) is unsatisfiable, then there is no actual run with signature  $V_0 \dots V_i$  such that  $D_k$  is an empty dataset. If one can reach node  $V_0$  with a condition that is satisfiable, and satisfied by  $D_{in}$ , then such a run exists. The main technical points to obtain decidability are : 1) show that one can effectively compute preconditions, and 2) that for all computed preconditions, satisfiability is decidable.

A First Order formula (in prenex normal form) over a set of variables  $X$  is a formula of the form  $\phi ::= P(X).\psi(X)$  where  $P(X)$  is an alternation of quantifiers and variable names in  $X$ , i.e. sentences of the form  $\forall x_1 \exists x_2, \dots$  called the *prefix* of  $\phi$  and  $\psi(X)$  is a quantifier free formula made of boolean combinations of atoms of the form  $R(x_1, \dots x_k)$ ,  $x_i = x_j$  called the *matrix* of  $\phi$ .  $R(x_1, \dots x_k)$  is a relational statement or a predicate. Each variable in  $X$  has its own domain. A variable assignment is a function  $\mu$  that associates a value from their domain to variables in  $X$ . A formula of the form  $\exists x, \phi(x)$  is true if and only if there is a way to choose a value for  $x$  such that  $\phi(x)$  is satisfied. A formula of the form  $\forall x, \phi(x)$  is true if and only if, for every possible choice of a value for  $x$ ,  $\phi(x)$  is satisfied.

Letting  $X_1 = \{x_1, \dots x_k\} \subseteq X$  we will often write  $\vec{\forall} X_1$  instead of  $\forall x_1. \forall x_2 \dots \forall x_k$ . Similarly, we will write  $\vec{\exists} X_1$  instead of  $\exists x_1. \exists x_2 \dots \exists x_k$ . Given an FO formula in prenex normal form, we will use w.l.o.g. formulas of the form  $\vec{\forall} X_1 \vec{\exists} X_2 \dots \psi(X)$  or  $\vec{\exists} X_1 \vec{\forall} X_2 \dots \psi(X)$ , where  $\psi(X)$  is quantifier free matrix, and for every  $i \neq j$ ,  $X_i \cap X_j = \emptyset$ . Every set of variables  $X_i$  is called a *block*. It is well known that satisfiability of first order logic is undecidable in general, but it is decidable for several fragments. In this work, we consider the *Separated Fragment (SF)* of FO, for which satisfiability of a formula is decidable [18]. The SF fragment is reasonably powerful and subsumes the *Monadic Fragment* [15] (where predicates can only be unary) and the *Bernays-Schonfinkel-Ramsey (BSR)* fragment of FO [3] (formulas of the form  $\vec{\exists} X_1 \vec{\forall} X_2 \dots \psi(X)$  where  $\psi(X)$  is quantifier-free).

Let  $A$  be an atom of a formula, and  $Vars(A)$  be the set of variables appearing in  $A$ . We say that two sets of variables  $Y, Z \subseteq X$  are separated in a quantifier free formula  $\phi(X)$  iff for every atom  $A$  of  $\phi(X)$ ,  $Vars(A) \cap Y = \emptyset$  or  $Vars(A) \cap Z = \emptyset$ . *Separated formulas* are formulas of the form  $\vec{\exists} Z, \vec{\forall} X_1 \vec{\exists} Y_2 \dots \vec{\forall} X_n \vec{\exists} Y_n \psi(X)$ , and the sets  $X_1 \cup \dots \cup X_n$  and  $Y_1 \cup \dots \cup Y_n$  are separated. Every separated formula can be rewritten into an equivalent formula in the BSR fragment, i.e. of the form  $\vec{\exists} U. \vec{\forall} V \psi'(U \cup V)$  (which yields decidability of satisfiability for SF formulas).

A well known decidable fragment is  $(FO^2)$ , that uses only two variables [16]. However, as this fragment forbids in particular atoms of arity greater than 2, which is a severe limitation when addressing properties of datasets. An interesting 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. It was demonstrated that  $FO^2BD$  formulas are closed under computation of weakest

preconditions for a set of simple SQL operations [8]. All the results demonstrated in our paper would hold in a  $FO^2BD$  setting. However, the setting that we propose has to handle datasets where fields are imprecise user inputs, that are better captured with unbounded domains. We will show hereafter that the separated fragment of FO suffices to encode emptiness of a dataset, and that all preconditions on contents of datasets built along an execution leading to a deadlock can also be encoded in SF. This will be of particular importance to prove decidability of termination.

**PROPOSITION 2.** *Let  $\psi_i$  be an FO formula. Then, for any move  $m_i$  from  $C_{i-1}$  to  $C_i$  and any formula  $\psi_i, \psi_{i-1} = wp[m_i]\psi_i$  is an effectively computable FO Formula. Further, if  $\psi_i$  is in separated form, then  $\psi_{i-1}$  is also in separated form.*

**PROOF.** For every FO Property  $\psi_i$ , and every type of move  $act_i$  we write a technical lemma that build formula  $\phi_{i-1} = wp[act_i]\psi_i$  that is the weakest precondition on input datasets for minimal nodes in  $W_{i-1}$  allowing property  $\psi_i$  to hold on datasets  $D_1, \dots D_K$  used as inputs of minimal nodes of  $W_i$ . FO is closed under computation of weakest precondition, and weakest preconditions of separated formulas are also separated. We refer interested readers to Appendix A.5 for these technical lemmas.  $\square$

**PROPOSITION 3.** *Let CW be a complex workflow. Given a signature  $V_0 \dots V_i$  in the execution tree of CW, and a separated FO formula  $\phi$  one can decide:*

- if there exists a run  $\rho$  with input dataset  $D_{in}$  and signature  $V_0 \dots V_i$  such that  $\phi$  holds at  $i^{th}$  step of  $\rho$ .
- if there exists a run  $\rho$  of CW with input dataset  $D_{in} \in \mathcal{D}_{in}$  with signature  $V_0 \dots V_i$  such that  $\phi$  holds at  $i^{th}$  step of  $\rho$  when  $\mathcal{D}_{in}$  is defined in separated FO.

**PROOF.** This is a straightforward consequence of Prop. 2. For a given path  $V_0 \xrightarrow{act_1} V_1 \dots \xrightarrow{act_k} V_k$  in the tree  $\mathcal{B}$  of a complex workflow and a formula  $\phi_k$ , there exists a sequence of moves  $C_0 \xrightarrow{act_1} C_1 \dots \xrightarrow{act_k} C_k$  where  $C_k \models \phi_k$  iff the sequence  $C_0 \xrightarrow{act_1} C_1 \dots \xrightarrow{act_{k-1}} C_{k-1}$  end in a configuration  $C_{k-1}$  such that  $C_{k-1} \models wp[act_k]\phi_k$  (by definition of weakest precondition). If  $wp[act_k]\phi_k$  is not satisfiable, then the move from  $C_{k-1}$  to  $C_k$  cannot produce datasets fulfilling  $\phi_k$ . One can decide whether  $wp[act_k]\phi_k$  is satisfiable, as  $\phi_k$  is in separated form, and by Prop. 2,  $wp[act_k]\phi_k$  is also separated.

Now, one can build inductively all weakest preconditions  $WP_{k-1}, WP_{k-2}, WP_0$  that have to be satisfied respectively by configurations  $C_{k-1}, \dots, C_0$ . If any of these preconditions is unsatisfiable, then there exists no run with signature  $V_0 \dots V_k$  leading to a configuration that satisfies  $\phi_k$ . Assume that  $WP_{k-1}, WP_{k-2}, \dots, WP_0$  are satisfiable. Then it remains to check that  $D_{in} \models WP_0$  to guarantee existence of a run with signature  $V_0 \dots V_k$  that starts with input data  $D_{in}$  and leads to a configuration that satisfies  $\phi_k$ . Similarly, if  $D_{in}$  is given as separated FO formula  $\phi_{in}$  then proving that a some  $D_{in} \in \mathcal{D}_{in}$  allows a run with signature  $V_0 \dots V_k$  leading to a configuration that satisfies  $\phi_k$  amounts to checking satisfiability of  $\phi_{in} \wedge WP_0$ .  $\square$

Now, emptiness of a dataset  $D_k$  with relational schema  $rs_k = (rn_k, A_k)$  can be encoded with the separated formula  $\phi_{D_k}^0 ::= \exists x_1, \dots, x_{|A_k|}, rn_k(x_1, \dots, x_{|A_k|}) \in D_k$ . This means that condition *iii*) in Proposition 1 can be effectively checked.

**THEOREM 4.5.** *Universal termination of complex workflows is decidable (for single and sets of inputs).*

**SKETCH.** Complex workflows terminate iff they have bounded recursive schemes, and if they do not deadlock. The first condition can be checked by considering how tasks are rewritten (see appendix for details). If  $CW$  allows no unbounded recursive scheme, its symbolic execution tree is finite. Then, we can detect deadlocks and potential deadlocks in this tree. If a deadlock exists, then the complex workflow does not terminate universally. Potential deadlocks occur only if a vertex  $V_i = (W_i, Ass_i, \mathbf{Dass}_i^S)$  in the tree is such that  $W_i$  contains an executable split node  $n_j$ . Now, emptiness of some dataset  $D_k \in \mathbf{Dass}_i^S$  with relational schema  $rs_k = (rn_k, A_k)$  can be encoded with a disjunction of separated formulas  $\psi_k^0$ . Hence, using Proposition 3, one can decide whether a run starting with input data  $D_{in}$  (or from some input data in  $\mathcal{D}_{in}$ ) which signature is the path leading to  $V_i$  such that  $D_k = \emptyset$  for some  $D_k$  used as input of a split node exists.  $\square$

The constructive proof of Proposition 3 immediately gives an algorithm to check existence of a deadlock during an execution of a workflow. First of all, assume that a path in  $\mathcal{B}$  exists from  $V_0$  to some vertex  $V_k$  where a dataset  $D_k$  has to be split. As explained in the proof of Theorem 4.5, the property that  $D_k = \emptyset$  is expressible in the separated fragment of FO, so one can check satisfiability of a sequence of weakest preconditions up to  $WP_0$  for every potential deadlock vertex in the tree. If none of the potential deadlocks allows to prove existence of a run leading to a configuration where an empty dataset has to be split, then in all execution, split actions can occur safely and never deadlock a run. Then, as all other operation have no precondition on the contents of datasets, if a path  $V_0 \dots V_{dead}$  to a deadlock vertex  $V_{dead}$  exists after verifying that potential deadlocks are harmless, then a run with signature  $V_0 \dots V_{dead}$  exists (for any input dataset). Algorithm 1 shows how to decide universal termination for a particular input  $D_{in}$ . This algorithm can be easily adapted to address termination for a set of inputs  $\mathcal{D}_{in}$ .

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 a dedicated logic such as LTL FO [4] (a logic that address both properties of data and runs) are also undecidable. However, one can notice that in the counter machine encoding in the proof of Theorem 4.2, instructions execution, require refinements, recursive schemes and split nodes (in particular to encode zero tests). So, infinite runs of a counter machine can be encoded only if rule 4 can be applied an infinite number of times. We hence slightly adapt the semantics of Section 3, and in particular rule  $R_4$ , and replace it by a *restrictive decomposition* ( $RD$ ) rule. Intuitively, the ( $RD$ )

---

**Algorithm 1:** Universal Termination Decision

---

**Data:** A complex workflow  
 $CW = (W_0, \mathcal{T}, T_{cs}, \mathcal{U}, sk, \mathcal{P})$   
**Result:** A verdict  $\in \{TERM, NO - TERM\}$

- 1 **If**  $CW$  has unbounded recursion **Return** NO-TERM
- 2 Build the symbolic execution tree  $\mathcal{B} = (V, E)$  of  $CW$
- 3  $V_{split} = \{(W, Ass, \mathbf{Dass}^S) \in V \mid W \text{ has splittable nodes}\}$
- 4 **for**  $v \in V_{split}$  **do**
- 5  $\rho_v = v_0 \xrightarrow{a_0} v_1 \dots \xrightarrow{a_{k-1}} v_k = v$  //path from  $v_0$  to  $v$
- 6 **for**  $n \in \text{split nodes of } \min(W_i)$  **do**
- 7  $WP ::= wp[a_{k-1}](\bigvee_{D_k \in \mathbf{Dass}^S(n)} D_k = \emptyset)$
- 8 **for**  $i = k - 1..1$  **do**
- 9 Check satisfiability of  $WP$
- 10 **if**  $WP$  not satisfiable **then**
- 11 | **break;** //unfeasible path
- 12 **end**
- 13  $WP ::= wp[a_{i-1}]WP$
- 14 **end**
- 15 //WP= WPO
- 16 **if**  $WP$  satisfiable  $\wedge D_{in} \models WP$  **then**
- 17 | **return** NO-TERM
- 18 **end**
- 19 **end**
- 20 **end**
- 21 //All Split nodes have non-empty input datasets
- 22 **if**  $\exists v \in V$  without successors and  $v$  is not final **then**
- 23 | **return** NO-TERM
- 24 **end**
- 25 **return** TERM

---

rule refines a task as in rule  $R_4$ , but forbids decomposing the same task an unbounded number of times.

**Rule 4'** (RESTRICTED TASK REFINEMENT): Let  $T = \{t_{int}, t_2, \dots, t_f\}$  be a set of tasks of size  $n$ . Let  $KD = (k_1, k_2, \dots, k_n) \in \mathbb{N}^n$  be a vector constraining the number of refinements of task  $t_i$  that can occur in a run  $\rho$ . In the context of crowdsourcing, this seems a reasonable restriction. Restrictive decomposition  $RD$  is an adaptation of rule  $R_4$  that fixes an upper bound  $k_i$  on the number of decomposition operations that can be applied for each task  $t_i$  in a run. We augment configurations with a vector  $S \in \mathbb{N}^n$ , such that  $S[i]$  memorizes the number of decompositions of task  $t_i$  that have occurred. Rules 1-3 leave counter values unchanged, and rule 4 becomes:

$$\begin{aligned}
& \exists n \in \min(W), \exists u = Ass(n), t_i = \lambda(n) \in T_{cx} \wedge S[i] \leq k_i \\
& \wedge \exists W_s = (N_s, \rightarrow_s, \lambda_s) \in Profile(t_i, u) \\
& \wedge Ass' = Ass \setminus \{(n, Ass(n))\} \wedge \mathbf{Dass}'(\min(W_s)) = \mathbf{Dass}(n) \\
& \wedge \forall x \in N_s \setminus \min(W_s), \mathbf{Dass}'(x) = \emptyset^{Pred(x)} \\
& \wedge W' = W_{[n/W_s]} \\
& \forall j \in 1 \dots |T|, S'[j] = S[j] + 1 \text{ if } j = i, S'[j] \text{ otherwise}
\end{aligned}$$


---


$$(W, Ass, \mathbf{Dass}, S) \xrightarrow{split(t_i)} (W', Ass', \mathbf{Dass}, S') \quad (5)$$

Following the *RD* semantics, each task  $t_i$  can be decomposed at most  $k_i$  times. To simplify notations, we choose a uniform bound  $k \in \mathbb{N}$  for all tasks, i.e.  $\forall i \in 1..n, k_i = k$ . However, all results established below extend to a non-uniform setting. We next show decidability of existential termination under the *RD* semantics. First, we give an upper bound on the length of runs under *RD* semantics. Let  $k$  be a uniform bound on the number of decompositions,  $CW = (W_0, \mathcal{T}, T_{cs}, \mathcal{U}, sk, \mathcal{P})$  be a complex workflow with a set of tasks of size  $n$ , and  $C_0 = (W, Ass_0, \mathbf{Dass}_0)$  be its initial configuration.

**PROPOSITION 4.** *Let  $\rho = C_0 \dots C_k$  be a run of a complex workflow under *RD* semantics. The length of  $\rho$  is bounded by  $L(n, k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$*

**SKETCH.** Configurations can only grow up to a size smaller than  $C(n, k) = k \cdot n^2 + |W_0|$  via rule R4, and rules R1-R3 can be applied only a finite number of times from each configuration.  $\square$

Under *RD* semantics, a symbolic execution tree  $\mathcal{B}$  is necessarily finite and of bounded depth. A run terminates iff it goes from the initial configuration to a final one. If such run exists, then there exists a path in  $\mathcal{B}$  from the initial vertex to a final vertex with signature  $\Pi = V_0 \dots V_n$ . Further, if this path visits a potential deadlock and executes a splitting task  $\lambda(n)$  for some split node  $n_j$ , then every dataset used as input of  $n_j$  must be non-empty. To show that this path is realizable, it suffices to show existence of a run with signature  $\Pi$  that ends in a configuration  $C_n$  satisfying property  $\phi ::= true$ . Proposition 3 shows how to compute backwards the weakest preconditions demonstrating existence of such run of *CW*. An immediate consequence is that existential termination of complex workflows is decidable under restricted decomposition semantics.

**THEOREM 4.6.** *Existential termination for a single input  $D_{in}$  or for a set of inputs  $\mathcal{D}_{in}$  are decidable under restricted decomposition semantics.*

## 5 PROPER TERMINATION

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  $D_{out}$ . We assume the client sees the crowdsourcing platform as a black box, and simply asks for the realization of a complex tasks that need specific competences. However, the client may have requirements on the type of output returned for a particular input. We express this constraint with a First Order 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. This is captured by the notion of *proper termination*.

**Definition 5.1 (Proper termination).** 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  $\mathcal{R}uns(CW, \mathcal{D}_{in})$  terminates properly if it ends in a final configuration and

returns a dataset  $D_{out}$  such that  $D_{in}, D_{out} \models \psi_{in,out}$ . *CW* terminates properly existentially with inputs  $\mathcal{D}_{in}$  iff there exists a run  $\mathcal{R}uns(CW, D_{in})$  for some  $D_{in} \in \mathcal{D}_{in}$  that terminates properly. *CW* terminates properly universally with inputs  $\mathcal{D}_{in}$  iff all runs in  $\mathcal{R}uns(CW, \mathcal{D}_{in})$  terminate properly for every  $D_{in} \in \mathcal{D}_{in}$ .

Proper termination guarantees completion of a whole workflow *CW*, and construction of an output dataset  $D_{out}$  satisfying the constraints imposed by a client. In general, termination of a run does not guarantee its proper termination. A terminated run may return a dataset  $D_{out}$  such that pair  $D_{in}, D_{out}$  does not comply with constraints  $\psi_{in,out}$  imposed by the client. For instance, a run may terminate with an empty dataset while the client asked for an output with 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. Proper termination can be immediately brought back to a termination question, by setting  $\psi_{in,out} = true$ . We hence have the following corollary.

**COROLLARY 5.2.** *Existential proper termination of a complex workflow is undecidable.*

In this section, we show that proper termination can be handled through symbolic manipulation of datasets, that give constraints on the range of possible values of record fields, and on the cardinality of datasets. We handle execution symbolically, i.e. we associate a symbolic data description to inputs of every node which task is executed, and propagate the constraints on this data to the output(s) produced by the execution of the task. AS for termination, weakest preconditions can be built. However, universal termination is decidable only with some restrictions on the fragment of FO used to constrain relation between inputs and outputs.

**THEOREM 5.3.** *Let *CW* be a complex workflow, and  $\psi_{in,out}$  be a constraint on inputs and outputs written in FO. Then:*

- *existential and universal proper termination of *CW* are undecidable, even under *RD* semantics.*
- *if  $\psi_{in,out}$  is in the separated fragment of FO, then*
  - *existential proper termination is decidable under *RD* semantics*
  - *universal proper termination is decidable.*

**PROOF.** Let us first prove the undecidability part: It is well known that satisfiability of *FO* is undecidable in general. One can take an example of formula  $\psi_{unsat}$  which satisfiability is not decidable. One can also build a formula  $\psi_{id}$  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, both under standard and *RD* semantics. Then,  $CW_{id}$  terminates properly iff there exists a dataset  $D_{in}$  such that  $D_{in} \models \psi_{unsat}$ , i.e. if  $\psi_{unsat}$  is satisfiable. Universal and existential proper termination are hence undecidable problems.

For the decidable cases, one can apply the technique of Theorem 4.6. One can build the symbolic execution tree, check that all runs terminate. Then for every terminated leaf  $n$  of the execution tree, one can compute a chain of weakest preconditions  $WP_0, WP_1, \dots, WP_n$  that have to be enforced to execute successfully  $CW$  and terminate in node  $n$ . In particular,  $WP_n ::= true$ . Then, one has to check satisfiability of  $\psi_{proper,n} ::= \bigwedge WP_i \wedge \psi_{in,out}$ . As all  $WP_i$ 's are in the separated fragment of FO, if  $\psi_{in,out}$  is separated, then so is  $\psi_{proper,n}$ . Hence, existential proper termination is decidable for complex workflows executed under restricted decomposition if  $\psi_{in,out}$  is expressed in separated FO.

Last, for universal proper termination, arguments of Theorem 4.5 apply: recursive schemes prevent termination, and if recursion is bounded, one can check for every path of the SET that no deadlock needs to occur, and that the weakest preconditions combined with  $\psi_{in,out}$  are satisfiable.  $\square$

At first sight, restricting to the separated fragment of FO can be seen as a limitation. However, the existential fragment of FO is already a very useful logic, that can express non-emptiness of outputs: property  $\exists x_1, \dots, \exists x_k, rn(x_1, \dots, x_k) \in D_{out}$  expresses the fact that the output should contain at least one record. Similarly, one can express properties to impose that every input has been processed. For instance, the property

$$\psi_{in,out}^{valid} ::= \forall x_1 \dots x_k, rn(x_1, \dots, x_k) \in D_{in} \\ 0.3cm \implies \exists y_1 \dots y_q, rn(x_1, \dots, x_k, y_1, \dots, y_q) \in D_{out}$$

Formula  $\psi_{in,out}^{valid}$  asks that every input in  $D_{in}$  is kept and augmented by additional information. This formula can be rewritten into another formula with a single alternation of quantifiers of the form

$$\forall x_1 \dots x_k, \exists y_1 \dots y_q, \neg rn(x_1, \dots, x_k) \in D_{in} \\ \vee rn(x_1 \dots x_k, y_1 \dots y_q) \in D_{out}$$

This latter formula is in BSR form, which is a subset of the separated fragment of FO.

## 6 CONCLUSION

We have proposed data centric workflows for crowdsourcing applications. The model includes a higher-order operation, that allows splitting of tasks and datasets to decompose a workflow into orchestration of simple basic tasks. This gives complex workflows a huge expressive power. On this model, universal termination is decidable. If requirements on inputs and outputs are expressed with separated FO, universal proper termination is decidable too. Existential termination is not decidable in general. With the reasonable assumption that tasks cannot be decomposed an arbitrary number of times, existential termination and existential proper termination (with separated FO requirements) are decidable.

Several question remain open: satisfiability of separated FO is non-elementary [18], but in the formulas defining weakest preconditions in termination problems, all variable blocks are separated. We conjecture that preconditions close to BSR formulas, which could yield NEXPTIME complexity for the termination problem. Beyond complexity issues, complex

workflows raise other problems such as synthesis of appropriate pricing (find incentives that maximize the probability of termination), or synthesis of schedulers to guarantee termination with appropriate user assignment. Other research directions deals 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 tradeoffs between performance and accuracy of answers...

## REFERENCES

- [1] ABITEBOUL, S., SEGOUFIN, L., AND VIANU, V. Static analysis of active XML systems. *Trans. Database Syst.* 34, 4 (2009), 23:1–23:44.
- [2] ABITEBOUL, S., AND VIANU, V. Collaborative data-driven workflows: think global, act local. In *Proc. of PODS'13* (2013), ACM, pp. 91–102.
- [3] BERNAYS, P., AND SCHÖNFINKEL, M. Zum entscheidungsproblem der mathematischen logik. *Mathematische Annalen* 99, 1 (1928), 342–372.
- [4] DAMAGGIO, E., DEUTSCH, A., AND VIANU, V. Artifact systems with data dependencies and arithmetic. *Trans. on Database Systems* 37, 3 (2012), 22.
- [5] DANIEL, F., KUCHERBAEV, P., CAPPIELLO, C., BENATALLAH, B., AND ALLAHBAKHSH, M. Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. *ACM Computing Surveys* 51, 1 (2018), 7.
- [6] DEUTSCH, A., SUI, L., VIANU, V., AND ZHOU, D. Verification of communicating data-driven web services. In *Proc. of PODS'06* (2006), ACM, pp. 90–99.
- [7] GARCIA-MOLINA, H., JOGLEKAR, M., MARCUS, A., PARAMESWARAN, A., AND VERROIOS, V. Challenges in data crowdsourcing. *Trans. on Knowledge and Data Engineering* 28, 4 (2016), 901–911.
- [8] ITZHAKY, S., KOTEK, T., RINETZKY, N., SAGIV, M., TAMIR, O., VEITH, H., AND ZULEGER, F. On the automated verification of web applications with embedded SQL. In *Proc. of ICDT'17* (2017), vol. 68 of *LIPICs*, pp. 16:1–16:18.
- [9] KITTUR, A., SMUS, B., KHAMKAR, S., AND KRAUT, R. Crowdforge: Crowdsourcing complex work. In *Proc. of UIST'11* (2011), ACM, pp. 43–52.
- [10] KOUTSOS, A., AND VIANU, V. Process-centric views of data-driven business artifacts. *Journal of Computer and System Sciences* 86 (2017), 82–107.
- [11] KUCHERBAEV, P., DANIEL, F., TRANQUILLINI, S., AND MARCHESE, M. Crowdsourcing processes: A survey of approaches and opportunities. *IEEE Internet Computing* 20, 2 (2016), 50–56.
- [12] KULKARNI, A., CAN, M., AND HARTMANN, B. Collaboratively crowdsourcing workflows with turkomatic. In *Proc. of CSCW'12* (2012), ACM, pp. 1003–1012.
- [13] LI, G., WANG, J., ZHENG, Y., AND FRANKLIN, M. Crowdsourced data management: A survey. *Trans. on Knowledge and Data Engineering* 28, 9 (2016), 2296–2319.
- [14] LITTLE, G., CHILTON, L., GOLDMAN, M., AND MILLER, R. Turkit: tools for iterative tasks on mechanical turk. In *Proc. of HCOMP'09* (2009), ACM, pp. 29–30.
- [15] LÖWENHEIM, L. Über möglichkeiten im relativkalkül. *Mathematische Annalen* 76, 4 (1915), 447–470.
- [16] MORTIMER, M. On languages with two variables. *Mathematical Logic Quarterly* 21, 1 (1975), 135–140.
- [17] NIGAM, A., AND CASWELL, N. Business artifacts: An approach to operational specification. *IBM Systems Journal* 42, 3 (2003), 428–445.
- [18] STURM, T., VOIGT, M., AND WEIDENBACH, C. Deciding first-order satisfiability when universal and existential variables are separated. In *Proc. of LICS '16* (2016), ACM, pp. 86–95.
- [19] VAN DER AALST, W., VAN HEE, K., TER HOFSTEDE, A., SIDOROVA, N., VERBEEK, H., VOORHOEVE, M., AND WYNN, M. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing* 23, 3 (2011), 333–363.
- [20] ZHENG, Q., WANG, W., YU, Y., PAN, M., AND SHI, X. Crowdsourcing complex task automatically by workflow technology. In *MiPAC'16 Workshop* (2016), Springer, pp. 17–30.

## A APPENDIX : PROOFS

### A.1 Proof of Theorem 4.2

**Theorem 4.2** Existential termination of complex workflows is an undecidable problem.

The proof is done by reduction from the halting problem of two counter machines to termination of complex workflows.

PROOF. 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 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  $inst_q = inc(q, c_i, q')$ , depicting the fact that the machine is in state  $q$ , increases the value of counter  $c_i$  by 1, and moves to a new state  $q'$ . Instructions in  $I_2$  are of the form  $inst_q = dec(q, c_i, q'')$ , depicting the fact that the machine is in state  $q$ , if  $c_i == 0$ , the machine moves to new state  $q'$  without making any change in the value of counter  $c_i$ , and otherwise, decrements the counter  $c_i$  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  $c_2$ .

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 reachability 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 a particular final state and  $n_1, n_2$  are arbitrary values of the counter. It is well known that this reachability problem is undecidable.

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

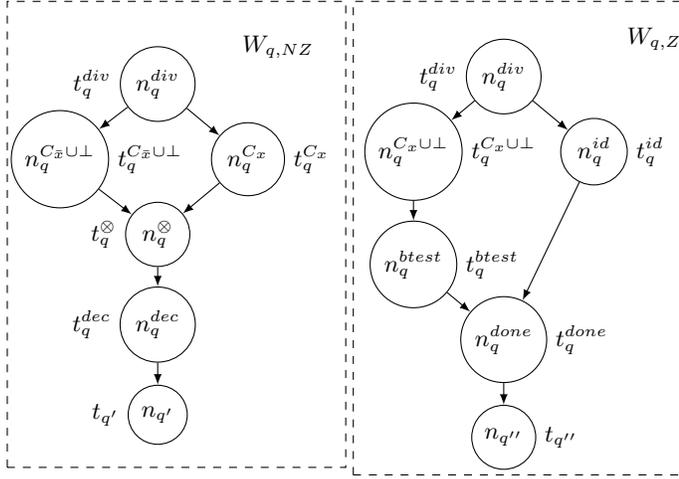
- We Consider a dataset  $D$  with relational schema  $rs = (R, \{k, cname\})$  where  $k$  is a unique identifier, and  $cname \in Cnt_1, Cnt_2, \perp$ . Clearly, we can encode the value of counter  $c_x$  with the cardinal of  $\{(k, n) \in D \mid n = Cnt_x\}$ . We start from a configuration where 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 user 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  $(n_q^{inc}, n_{q'}) \in \rightarrow$ ,  $\lambda(n_q^{inc}) = t_q^{inc}$  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  $dec(q, c_x, q'')$ , we create a complex task  $t_q$  and a worker  $u_q$  who can

choose to refine  $t_q$  according to profiles  $Profile(t_q, u_q) = \{W_{q,Z}, W_{q,NZ}\}$ . 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 workflow in a profile is non-deterministic, worker  $u_q$  can choose one or the other.

- Let us now detail the contents of  $W_{q,NZ}$ . This workflow is composed of nodes  $n_q^{div}, n_q^{C_x}, n_q^{C_x \cup \perp}, n_q^{\otimes}, n_q^{dec}$  and  $n_{q'}$ , respectively labeled by tasks  $t_q^{div}, t_q^{C_x}, t_q^{C_x \cup \perp}, t_q^{\otimes}, t_q^{dec}$  and  $t_{q'}$ . The dependence relation in  $W_{q,NZ}$  contains pairs  $(n_q^{div}, n_q^{C_x}), (n_q^{div}, n_q^{C_x \cup \perp}), (n_q^{C_x}, n_q^{\otimes}), (n_q^{C_x \cup \perp}, n_q^{\otimes}), (n_q^{\otimes}, n_q^{dec})$  and  $(n_q^{dec}, n_{q'})$ . The role of  $t_q^{div}$  is to split  $\mathbf{Dass}(n_q^{div})$  into to disjoint parts: the first one contains records of the form  $R(k, C_x)$  and the second part all other records. Tasks  $t_q^{C_x}$  and  $t_q^{C_x \cup \perp}$  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 form  $R(k, C_x)$ , 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 is 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 is 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 workflow 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, here  $C_k = (q, v_1, v_2)$  there exists a single non-deadlocked run, and that this run terminates of configuration  $(W, ass, \mathbf{Dass})$  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.



**Figure 2: Encoding of Non-zero test followed by decrement (left), and Zero Test followed by state change (right).**

□

## A.2 Proof of Proposition 1

LEMMA A.1. *Let  $\mathcal{B}$  be a tree with a potential deadlock  $V_i$  with successors  $v_{i,1}, \dots, v_{i,k}$  corresponding respectively to splitting of nodes  $n_1, \dots, n_k$  in the workflow part of node  $V_i$ . Then a run  $\Pi$  with signature  $V_0 \dots V_i$  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  $C_i$  is labeled by a complex task and is already assigned a competent user, then assignment of this node and input data will not change in any successor during execution. So, if this node cannot split and distribute data due to the fact that  $D_k = \emptyset$ , it will never be able to split this data later in an execution that starts with prefix that has signature  $\Pi$ . □

This lemma has useful consequences: for a potential deadlock, it is sufficient to detect that one input dataset  $D_{n_j}$  for a split node is empty to claim that there exists an execution with a signature that has  $V_0 \dots V_i$  as prefix, and that deadlocks.

**Proposition 1:** A complex workflow terminates universally iff the following conditions hold:

- i) Its symbolic execution tree is finite (there is no unbounded recursion)
- ii) there exists no path  $V_0 \dots V_i$  in the symbolic execution tree such that  $V_i$  is a deadlock
- iii) there exists no run with signature  $V_0 \dots V_i$  where  $V_i$  is a potential deadlock, with  $D_k = \emptyset$  for some  $D_k \in \mathbf{Dass}(n_j)$  and for some minimal split node  $n_j$  of  $W_i$ .

PROOF. First, notice that all runs of a complex workflow have their signature in the Symbolic execution tree, as application of a rule never considers data contents, but only the structure of a workflow. Hence, even when some execution of a splitting task could be prevented by empty inputs, the symbolic execution tree contains edges symbolizing the effects of this splitting action on the workflow.

If all runs of a complex workflow  $CW$  terminate, then  $CW$  has no infinite run and no deadlocked run. As a consequence, its symbolic execution tree is finite, and contains no deadlock nodes. As executions of  $CW$  never meets deadlocks, one cannot find a run with signature  $V_0 \dots V_i$  where  $V_i = (W_i, Ass_i, \mathbf{Dass}_i^S)$  and is such that  $W_i$  has a minimal split node with an empty input dataset. Hence conditions i), ii), iii) are met.

Let us now assume that  $CW$  does not terminate. It means that this complex workflow either allows unbounded runs, or reaches deadlocks. If  $CW$  has an unbounded run  $\rho_\omega$ , then the workflow allows an unbounded recursive schemes, i.e. situations where successive refinement of a node  $n$  labeled by a task  $t$  leads to replace  $n$  by a subgraph that still contains a node  $n'$  with task  $t$ . Further, as  $\rho_\omega$  is an effective execution of  $CW$ , every rule applied in the execution of this run also applies during the construction of a symbolic execution tree, and hence this tree contains an infinite path (which violates condition i). If an execution of  $CW$  ends in a deadlocked configuration, then it means that either no rule applies from this configuration, or that the only next possible action is the execution of a split node that cannot be performed due to an empty input dataset. As nodes of the symbolic execution tree only differ from real configurations with their data, the first case means that the symbolic execution tree also contains a deadlock node from which no semantic rule applies (hence violating condition ii). For the second case, the deadlocked run ends in a configuration  $C_i$ . It has a signature  $V_0 \dots V_i$ , and there exists a input dataset  $D = \emptyset$  that prevents a minimal node from being executed (hence violating condition iii). □

## A.3 Proof of Theorem 4.5

**Theorem 4.5 :** Universal termination of a complex workflow is a decidable problem.

PROOF. First we can show that complex workflows terminate only if they have bounded recursive schemes, and do not deadlock. Let us assume that a complex workflow has unbounded recursive schemes, and that none of the task executions or refinement is ever deadlocked. Then, there exists a task  $t$  and an infinite run  $\rho = \rho_1.\rho_2 \dots$  such that every  $\rho_i$  terminates with a refinement of task  $t$ . Under the assumption that the system does not deadlock during this infinite runs, such an infinite recursive scheme occurs only if  $t$  can be rewritten through successive refinement steps into a workflow that contains a new occurrence of task  $t$ . This can be checked from the list of tasks and profiles. We build a graph  $RG = (\mathcal{T}, \rightarrow_{\mathcal{T}}, T_0)$  where  $T_0$  is the set of tasks that appear in  $W_0$ ,  $(t, t') \in \rightarrow_{\mathcal{T}}$  iff there exists a worker  $u$ , a workflow

$W_t = (N_t, \rightarrow_t, \lambda_t)$  in  $\mathcal{P}(t, u)$  and a node  $n \in N_t$  such that  $\lambda(n) = t'$ . An edge  $(t, t')$  means that one can rewrite  $t$  into a workflow that contains  $t'$ . If  $RG$  contains a cycle that is accessible from  $T_0$ , then the complex workflow contains a recursive scheme. If an infinite runs containing an infinite number of rewritings does not deadlock, then the workflow does not terminate. If all runs that unfold such a recursive scheme deadlock at some point, then the complex workflow does not terminate either. It remains to consider the case of complex workflows without unbounded recursive schemes. The executions of such workflows are of bounded length, and the complex workflow terminates (universally) iff all of them terminate.

Complex workflows terminate only if they have bounded recursive schemes, and if they do not deadlock. The former can be checked by considering how tasks are rewritten. If there is no unbounded recursive scheme allowed by  $CW$ , then its symbolic execution tree is finite. Then, we can detect deadlocks and potential deadlocks in this tree. If a deadlock exists, then the complex workflow does not terminate universally. Potential deadlocks occur only if a vertex  $V_i$  in the tree allows a move that is a split of a dataset  $D$ . Now, emptiness of a dataset  $D$  with signature  $rn = (rn, A)$  with  $A = (a_1, \dots, a_k)$  can be encoded with the separated formula of the form  $\exists x_1, \dots, x_k, rn(x_1, \dots, x_k) \in D$ . Hence, using Proposition 3, one can decide whether a run starting with input data  $D_{in}$  (or from some input data in  $\mathcal{D}_{in}$  which signature is the path leading to  $V_i$  such that  $D = \emptyset$  exists. This completes the proof of Theorem 4.5.  $\square$

#### A.4 Proof of Proposition 4

**PROPOSITION 5.** *Let  $\rho = C_0.C_1 \dots C_q$  be a run under RD semantics. Then, for every  $C_i = (W_i, Ass_i, D_{ass}_i)$ , the number of nodes in  $W_i$  is smaller than  $C(n, k) = k \cdot n^2 + |W_0|$ .*

**PROOF.** Each decomposition of a task  $t_i$  replaces a single node  $n$  by a new workflow with at most  $d_i = \max_{u \in \mathcal{U}} \max\{|W_j| \mid W_j \in Profile(t_i, u)\}$  nodes. Recall that decomposition profiles are known, and that all nodes of workflows in profiles are attached distinct task names. So, we have  $d_i < n$ . Every run  $\rho$  starting from  $C_0$  is a sequence of rule applications. Rule 1 does not affect the size of workflows in configurations, and rules 2 and 3 remove at most one node from the current workflow when applied. For each task  $t_i$ , a run  $\rho$  contains at most  $k$  occurrence of rule 4 refining a task of type  $t_i$ . Application of rule 4 to task  $t_i$  adds at most  $d_i$  nodes to the current workflow, and removes the refined node. All other rules decrease the number of nodes. One can notice that as each task can be decomposed at most  $k$  times, rule 4 can be applied at most  $k \cdot n$  times in a run following the RD semantics, even if this run is of length greater than  $k \cdot n$ . Let  $S_0 = |W_0|$ ,  $S_1 = S_0 + n - 1$ , and  $S_{i+1} = S_i + (n - 1)$ . For a fixed  $n$  and a fixed  $k$ , the maximal size of the workflow component  $W_i$  in every configuration  $C_i$  of a run under RD semantics is smaller than  $S_{k \cdot n} = |W_0| + (k \cdot n)(n - 1) = |W_0| + k \cdot n^2 - k \cdot n$ .  $\square$

**Proposition 4:** Let  $\rho = C_0 \dots C_k$  be a run of a complex workflow under RD semantics. The length of  $\rho$  is bounded by  $L(n, k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$

**PROOF.** Recall that a configuration is a triple  $C_i = (W_i, Ass_i, D_{ass}_i)$ , with  $Ass(n) = u_i$ . Each configuration is a "global state" of the execution of a complex workflow.  $W_i$  represents the work that needs to be done before completion,  $Ass$  the users assignment, and  $D_{ass}$  the data assignment. Recall that a configuration with a single node is necessarily a final configuration with a node  $n_f$  which task is to return all computed values during the execution of the complex workflow.

The only way to change user or data assignment part of configurations is to execute the task attached to a node (i.e., apply rule R2 or R3) or refine a node (i.e. apply rule 4). Starting from a configuration  $C_i$ , the maximal number of user assignment that can be performed is  $|W_i|$ , and along the whole run, as each node can be assigned an user at most once, the maximal number of applications of rule R1 is  $C(n, k)$ .

The length of a run  $\rho$  is  $|\rho| = |\rho|_1 + |\rho|_2 + |\rho|_3 + |\rho|_4$  where  $|\rho|_i$  denotes the number of applications of rule  $R_i$ . Now,  $|\rho|_1 \leq C(n, k)$ . Similarly,  $|\rho|_2 = |\rho|_3 + |\rho|_4$ . Last, rule R3 can be applied only a number of times bounded by the maximum number of created nodes, i.e.,  $|\rho|_3 \leq C(n, k)$ . So overall,  $|\rho| = |\rho|_1 + (|\rho|_2 + |\rho|_4) + |\rho|_3 \leq C(n, k) + C(n, k) + C(n, k)$ . Hence, the length of  $\rho$  is bounded by  $L(n, k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$ .  $\square$

#### A.5 Proof of Proposition 2

**Proposition 2:** Let  $\psi_i$  be an FO formula. Then, for any move  $m_i$  from  $C_{i-1}$  to  $C_i$  and any formula  $\psi_i, \psi_{i-1} = wp[m_i]\psi_i$  is an effectively computable FO Formula. Further, if  $\psi_i$  is in separated form, then  $\psi_{i-1}$  is also in separated form.

**PROOF.** Each move  $m_i$  in the execution tree represents a configuration change, and transforms input datasets  $D_1, \dots, D_n$  into output datasets  $D'_1 \dots D'_p$ . These transformations are projections, selections, joins, field addition update or enlargement, or a one to one automated linear transformation of records in a dataset. Slightly abusing the notation for FO used so far, we will write  $D_1, \dots, D_k \models \psi$  to denote that a set of datasets satisfies formula  $\psi$ . In the formula, given a relation  $rn(v_1, \dots, v_m)$  depicting a record in a dataset, we will also make clear in the formula which dataset contains the record, using a notation of the form  $rn(v_1, \dots, v_m) \in D_i$ . Note that this can still be expressed in FO, as one can equivalently work with a single global dataset  $DU$  and a unique relational schema  $rs_u$  containing all fields appearing in a dataset used in the workflow, and add a new field  $dnum$  indicating, for each record  $r$ , to which dataset this record belongs. With a global relational scheme, instead of writing  $D_1, \dots, D_k \models \exists w_1, \dots, w_p, rn(w_1 \dots w_p) \in D_3 \wedge \phi$ , one would write  $DU \models \exists w_1, \dots, w_p, nbrn_u(w_1 \dots w_p, nb) \wedge (nb = 3) \wedge \phi$ .

Given a transformation  $tr$  that transforms inputs  $D_1, \dots, D_n$  into outputs  $D'_1, \dots, D'_k$ , and an FO property  $\psi_{post}$ , the *weakest precondition* on  $D_1, \dots, D_n$  such that  $D'_1, \dots, D'_k \models \psi_{post}$  after execution of  $tr$  is an FO property  $\psi_{pre}$  such that  $D_1, \dots, D_n \models \psi_{pre}$  implies that  $D'_1, \dots, D'_k \models \psi_{post}$ . We will

write  $\psi_{pre}wp[tr]D'_1, \dots, D'_k \models \psi_{post}$  to denote the fact that  $\psi_{pre}$  is this weakest precondition, or simply  $\psi_{pre}wp[tr]\psi_{post}$  when outputs are clear from the context. Now, moves from one configuration to another are atomic actions that may involve several successive transformations. Similarly, given a move  $mv$  from a configuration  $C$  to a configuration  $C'$ , we write  $\psi_{pre}wp[mv]\psi_{post}$  to denote that  $\psi_{pre}$  is the precondition on datasets in use in  $C$  required for  $\psi_{post}$  to hold in datasets in use in  $C'$  after move  $mv$ . First, for each type of basic transformation  $tr$ , we give the weakest precondition of any FO formula  $\psi_{post}$ . We then build preconditions for atomic moves from them, to prove that all preconditions needed to reach deadlocks can be expressed as FO properties. For each transformation, we also show that separated FO formulas also give separated weakest preconditions. For a formula  $\psi$ , we denote by  $Vars(\psi)$  the variables  $x_1 \dots x_n$  used in  $\psi$ . In the rest of the proof, we assume that all formulas over variables  $x_1, \dots, x_n$  are in prenex normal form, and more precisely are of the form  $\psi ::= Q(Vars(\phi)), \phi$ , where  $Q(Vars(\psi))$  is the prefix (a string of variables and quantifiers  $\forall, \exists$ ),  $\phi$  is a boolean combination of quantifier free FO statements called the *matrix*. It includes relational statements of the form  $rn(x_1, \dots, x_k) \in D_j$  to describe the fact a record of the form  $rn(w_i, \dots, w_{i+p})$  belongs to dataset  $D_j$ , predicates indicating constraints on values of variables, such as  $x_1 \leq x_2$ , and equalities. Note that computing a weakest precondition for a transformation that impacts the contents of a dataset  $D_i$  when  $D_i \models \psi$  yields properties that should be satisfied *jointly* by several datasets  $D_1, \dots, D_q$  used to forge the contents of  $D_i$ , and that these properties cannot be necessarily considered as independent preconditions of the form  $D_1 \models \psi_1, \dots, D_q \models \psi_q$ . As datasets contents and properties are not independent, we will write *global properties* of datasets used by all minimal nodes in configurations under the form  $D_1, \dots, D_k \models \psi$ . We denote by  $RS(\psi)$  the set of relational statements used in  $\psi$ . We can now provide a series of lemmas proving that for each type of action, weakest preconditions are effectively computable, and transform separated FO formulas into separated FO formulas.

**LEMMA A.2 (WEAKEST PRECONDITION FOR PROJECTION).** *Let  $\phi$  be a FO formula, and  $act$  be an atomic action that projects the contents of some datasets. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  is a separated FO formula, then  $\psi$  is also separated.*

Let us assume that  $D'_1, \dots, D'_k \models \psi_{post}$  and that  $D'_j$  is a dataset with relational schema  $rs_j = (rn_j, A_j)$ , obtained by projection of some input dataset  $D_i$  with relational schema  $rs_i = (rn_i, A_i)$  on a subset of its fields. We have  $A_j \subseteq A_i$ , and letting  $A_i = (a_1, \dots, a_k)$ ,  $A_j$  is of the form  $(a_{i_1}, a_{i_2}, \dots, a_{i_q})$ . Clearly, if a FO formula  $\psi$  addresses values of attributes  $(a_{i_1}, a_{i_2}, \dots, a_{i_q})$  of records in relational schema  $rs_j$ , if a record  $r = (v_1, \dots, v_q)$  satisfies  $\psi$  and is obtained by projection of a record  $r' = (v'_1, \dots, v'_k)$  with relational schema  $rs_i$ , then  $r'$  also satisfies  $\psi$ . Similar reasoning holds when contains several instances of  $rs_j$ . Let  $RS(\psi_{post})$  contain  $KP$  instances of relational schema  $rs_j$ .

Then, we can replace each instance of  $rn_j(x_i, \dots, x_{i+q})$  by an instance of  $rn_i(x_i, \dots, x_{i+q}, y_{i+q+1}, y_{i+k})$ , where  $y_i$ 's are new variables addressing values of fields in  $A_i \setminus A_j$ . We denote by  $\psi_{post_{[rs_j/rs_i]}}$  the formula  $\psi_{post}$  where every instance of  $rs_j$  has been replaced this way. Similarly, letting  $Y = \{y_{i+q+1}, \dots, y_{i+k} \mid i \in 1..KP\}$ , we denote by  $Q'(Vars(\psi) \cup Y)$  the sentence  $Q(Vars(\psi)).Q_Y$  where  $Q_Y = q_1.y_1 \dots q_n.y_n$  is a sentence where  $y_i$ 's are all variables of  $Y$ ,  $q_i$ 's their quantifiers, and that associates existential quantifiers to variables of  $Y$  appearing in statements of the form  $rn_i(\dots)$  and universal quantifiers to variables of  $Y$  appearing in statements of the form  $\neg rn_i(\dots)$ . The weakest precondition for projection is hence a precondition on  $D'_1, \dots, D'_k$ , given as  $wp[Proj]\psi_{post} = Q'(Vars(\psi) \cup Y), \psi_{post_{[rs_j/rs_i]}}$

Clearly, as variables are added to increase the number of fields in relational statements, if  $\psi_{post}$  is separated,  $wp[Proj]\psi_{post}$  is also separated.

**LEMMA A.3 (WEAKEST PRECONDITION FOR R2R TRANSFORMATIONS).** *Let  $\phi$  be a FO formula, and  $act$  be an atomic action that transforms each record in a dataset into another record. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  is a separated FO formula, then  $\psi$  is also separated.*

**PROOF. Record to Record Transformation (R2R)** converts each record from an input dataset  $D_i$  to a new record corresponding to output dataset  $D_j$  by applying some linear transformation. Consider the dataset  $D_j$  with relational schema  $rs_j = (rn_j, B)$ . Let  $\psi_{post}$  be an FO formula such that  $D'_1, \dots, D'_j, \dots, D'_k \models \psi_{post}$ , and where  $D_j$  is obtained by R2R transformation of an input dataset  $D_i$  with relational schema  $rs_i(rn_i, A)$ . Let  $A = (a_1, \dots, a_p)$  and  $B = (b_1, \dots, b_q)$ . An R2R transformation from  $rs_i$  to  $rs_j$  is a transformation, that associates to each records  $r_1 = (v_1, \dots, v_p)$  with relational schema  $rs_i$ , where  $v_k$  is the value of attribute  $a_k$ , a record  $r_2 = (w_1, \dots, w_q)$  with relational schema  $rs_2$  such that every  $w_j$  is the value of attribute  $b_j$  obtained as a combination of values  $v_1, \dots, v_p$ . If  $v_1, \dots, v_p$  are numerical values then each  $w_j$  is a linear combination of the form  $w_j = k_{j,1}v_1 + k_{j,2}v_2 + \dots + k_{j,p}v_p + k_j$ . where  $k, k_1, \dots, k_p$  are constant values. This type of transformation allows to define mean values, sums of values in fields, etc...

Let  $\psi_{post}$  constrain values of variables  $W = w_1, \dots, w_h$ . Variables in  $W$  depict values of attributes  $b_1, \dots, b_q$  in records of  $D_j$  (i.e. they appear in a subformula of the form  $rn_j(w_1, \dots, w_q)$ ). Let  $Att(w_i)$  denote the attribute of variable  $w_i$ . We assume that the variables used under the scope of two subformulas of the form  $rn_2(w_1, \dots, w_q)$  and  $rn_2(w'_1, \dots, w'_q)$  are disjoint, and that equality of values is achieved through side formulas of the form  $w_i = w'_j$ . Note that even if transformation  $f$  is a record to record transformation, formula  $\psi_{post}$  can address values of more than one record, i.e. be of the form  $\exists w_1, \dots, w_q, w_{q+1} \dots w_{2,q}, rn_j(w_1, \dots, w_q) \wedge rn_j(w_{q+1}, \dots, w_{2,q}) \wedge \dots \phi$ . However, every record  $rn_j(w_1, \dots, w_q)$  is obtained as transformation of a record of the form  $rn_i(v_1, \dots, v_p)$ . Let  $K_{rn_j}$  be the number of subformulas of the form

$rn_j(w_i, \dots, w_{x+q})$  in  $\psi_{post}$ . We denote by  $\psi_{post[B/R2R(A)]}$  the formula  $\psi_{post}$  where every instance of relational schema  $rs_j$  i.e., an instance of the form  $rn_i(w_{k.q+1}, \dots, v_{k.(q+1)})$  is replaced by an instance  $rn_i(v_{k.p+1}, \dots, v_{k.(p+1)})$  of schema  $rs_i$ , and every occurrence of a variable  $w_i$  used in an instance of  $rs_j$  and appearing outside a relation is replaced by the linear combination of values  $v_{j+1} \dots v_{j+p}$  and constant  $k_j$  (where  $j = \lfloor \frac{i}{p} \rfloor$ ) allowing to obtain the value of variable  $w_i$ . Hence, the weakest precondition for R2R transformation is a precondition on  $D'_1, \dots, D_i, \dots, D'_k$ , given as

$$wp[R2R]\psi_{post} = \psi_{post[B/R2R(A)]}$$

One can notice that  $wp[R2R]\psi_{post}$  simply replaces atoms in a separated formula by other atoms, over new sets of variables. However, this transformation replaces a universally (resp. existentially) quantified block of variables by a new universally (resp. existentially) quantified block, which preserves vacuity of intersection of existential and universal variables. Hence, if  $\psi_{post}$  is separated, then  $wp[R2R]\psi_{post}$  is also separated.  $\square$

LEMMA A.4 (WEAKEST PRECONDITION FOR SELECTION OF RECORDS). *Let  $\phi$  be a FO formula, and act be an atomic action that selects records that satisfy a predicate  $P$  from datasets. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  and  $P$  are separated FO formulas, then  $\psi$  is also separated.*

PROOF. Let  $D'_1, \dots, D'_j, \dots, D'_k \models \psi_{post}$ , 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 with relational schema  $rs$ ). That is, the record selected from  $D_i$  by  $P$  are records that satisfy  $\psi_{sel} = \exists v_1, \dots, v_k, rn(v_1, \dots, v_p) \wedge P(v_1, \dots, v_k)$ .

Formula  $\psi_{post}$  is a formula of the form  $Q(Vars(\psi_{post})), \phi$ . 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, as for R2R transformation, we assume without loss of generality that these subformulas are over disjoint sets of variables. Let  $\phi_{rn,1}, \dots, \phi_{rn,K_{rn}}$  be the subformulas of  $\phi$  addressing tuples with relational schemas in  $rs$ . For  $i \in 1..K_{rn}$ , we let  $\phi_{rn,i}^P$  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. Last, let us denote by  $\phi_{\{\phi_{rn,i}\}\{\phi_{rn,i}^P\}}$  the formula where every  $\{\phi_{rn,i}\}$  is replaced by  $\{\phi_{rn,i}^P\}$ . The weakest precondition on  $D'_1, \dots, D_i, \dots, D'_k$  for a selection operation with predicate  $P$  is defined as

$$wp[Selection(\psi_{sel})]\psi_{post} = Q(Vars(\psi_{post})), \phi_{\{\phi_{rn,i}\}\{\phi_{rn,i}^P\}}$$

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  $\psi_{post}$ , then they remain existentially (resp. universally quantified). Hence, if  $\psi_{post}$  is separated, then  $wp[Selection(\psi_{sel})]\psi_{post}$  is also separated.  $\square$

LEMMA A.5 (WEAKEST PRECONDITION FOR FIELD ADDITION). *Let  $\phi$  be a FO formula, and act be an atomic action that adds new fields to a dataset. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  is a separated FO formulas, then  $\psi$  is also separated.*

PROOF. The **Field Addition** action adds an extra field to an existing relational schema, and populates this field. This transformation models entry of new information by users for each record in a dataset (for instance a tagging operation). Let  $D'_1, \dots, D'_j, \dots, D'_k \models \psi_{post}$ , and let  $D'_j$  be the modified dataset with relational schema  $rs(rn, A)$ . We let  $D_j$  be a dataset over relational schema  $rs_j = (rn_j, A_j)$ , where  $A_j = (a_1, \dots, a_p)$ . As  $D_j$  is obtained by adding a field to  $D_i$ , we have  $A_i = (a_1, \dots, a_{p-1})$ . We assume that constrains on possible values of new fields are provided by a predicate  $P_{add}(v_1, \dots, v_p)$  that is true if, value  $v_p$  is a legal value for field  $a_p$  if  $a_1, \dots, a_{p-1}$  take values  $v_1, \dots, v_p$  (if the value of field  $a_p$  can be any value in its domain, this predicate is simply *true*). Let  $K_{fld}$  be the number of subformulas of the form  $rn_j(\dots)$  or  $\neg rn(\dots)$  in  $\psi_{post}$  (again these subformulas are over disjoint variables). Formula  $\psi_{post}$  is hence a formula over variables  $W = Vars(\phi_{post})$  that contain at least a set of variables  $w_1, \dots, w_p, w_{p+1} \dots w_{K_{fld} \cdot p}$  appearing in relational subformulas.  $\psi_{post}$  is of the form  $Q(Vars(\psi_{post})), \phi$ , where  $\phi$  is a boolean combination of relational statements and comparisons of field values. Here, we can transform  $\phi$  over variables  $W$  into another formula  $\phi_{[rn_j|rn_i]}$ , where every relation statement of the form  $rn_j(w_k, \dots, w_{p+k})$  is replaced by a subformula  $rn_i(w_k, \dots, w_{p+k-1}) \wedge P_{add}(w_k, \dots, w_{p+k})$  in positive subformulas, and by a subformula of the form  $\neg(rn_i(w_k, \dots, w_{p+k-1}) \wedge P_{add}(w_k, \dots, w_{p+k-1}))$  otherwise.

The weakest precondition for the addition of a field  $a_p$  hence becomes:

$$wp[FA(a_p)]\psi_{post} : Q(Vars(\psi_{post})), \phi_{[rn_j|rn_i]}$$

Let us assume that  $\psi_{post}$  is separated. Then, as for he selection case,  $wp[FA(a_p)]\psi_{post}$  performs a syntactic replacement of a separated atom  $rn_j(w_k, \dots, w_{p+k})$  by a conjunction of separated atoms  $rn_i(w_k, \dots, w_{p+k-1}) \wedge P_{add}(w_k, \dots, w_{p+k})$ . Hence  $wp[FA(a_p)]\psi_{post}$  is also separated.  $\square$

LEMMA A.6 (WEAKEST PRECONDITION FOR FIELD ENLARGEMENT). *Let  $\phi$  be a FO formula, and act be an atomic action that selects records that adds imprecision to the contents of a field in dataset. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  and  $P$  are separated FO formulas, then  $\psi$  is also separated.*

PROOF. **Enlargement of field** is used to model the fact that users answers are sometimes subject to imprecision. The effect of imprecision is to replace a value in some field of a particular dataset with continuous domain by another value

that is close to the original value, i.e. at some distance  $\delta$ . Let  $D'_j$  be an output dataset with relational schema  $rs(rn, A)$  where  $A = (a_1, \dots, a_p)$ , and obtained by making a particular field  $a_j$  in an input dataset  $D_i$  with the same relational schema imprecise. Enlargement of field function transforms every input record  $r_1 = (v_1, \dots, v_j, \dots, v_p)$  where each  $v_i$  is the value of  $a_i$  to new record  $r_2 = (v_1, \dots, v'_j, \dots, v_p)$  such that  $v_j \in [v_j - \delta, v_j + \delta] \cap \text{Dom}(a_j)$ . One can notice that enlargement preserves the relational schema of input dataset  $D_i$ .

Let  $\psi_{post}$  be a FO property over a set of variables  $W$  and let  $D'_1, \dots, D'_j \dots D'_k \models \psi_{post}$ . If  $K_{rn}$  is the number of subformulas of the form  $rn(x_1, \dots, x_p) \in D'_j$ , then we can define  $\psi_{post}$  as a formula over  $V = W \cup Y$  where  $W = w_1, \dots, w_p, w_{p+1} \dots w_{K_{rn} \cdot p}$  is the set of variables used in these relational statements and  $Y$  the other variables.  $\psi_{post}$  is hence of the form  $\psi_{post} = Q(V), \phi_{post}$ .

The weakest precondition required so that  $D'_1, \dots, D'_j \dots D'_k \models \psi_{post}$  is a condition on values of variables in  $W$  such that, even after adding some imprecision to values of variables in set  $W_{imp} = w_j, w_{j+p}, \dots, w_{(K_{rn}-1) \cdot p+j}$   $\psi_{post}$  still holds. The weakest precondition hence includes the amount of imprecision on each variable in  $W_{imp}$ , and can be modeled by adding variables  $X = \{x_1, \dots, x_{K_{rn}}\}$  with domain  $[-\delta, +\delta]$  to existentially quantified variables. Note that adding imprecision to an universally quantified variables  $v$  is not needed, as the considered properties should hold for all possible values of  $v$  in its domain. Considering an expression of the form  $expr ::= k_1.w_1 + k_2.w_2 \dots k_p.w_p + k$ , the expression  $expr[v_j/v'_j + x_j]$  is obtained by replacing existentially quantified variable  $v_j$  by  $(v'_j + x_j)$  in  $expr$ . For a subformula of the form  $\phi = expr_1 \bowtie expr_2$ , where  $expr_1$  contains a variable  $v_{j+n.p}$  and  $expr_2$  contains a variable  $v_{j+m.p}$ , we denote by  $\phi^{imp} = expr_1[v_{j+n.p}/v_{j+n.p}+x_{j+n.p}] \bowtie expr_2[v_{j+m.p}/v_{j+m.p}+x_{j+m.p}]$  the formula where each occurrence of imprecise variable  $v_{j+n.p}$  is replaced by  $v_{j+n.p} + x_{j+n.p}$ . For a subformula of the form  $\phi = rn(v_1, \dots, v_j, \dots, v_k)$  where  $v_j$  is an existential variable corresponding to the enlarged field  $a_j$ ,  $\phi^{imp} = rn(v_1, \dots, v_j + x_j, \dots, v_k)$ . For formulas containing no existentially quantified enlarged variables,  $\phi^{imp} = \phi$ . Last, for formulas that are boolean combinations of subformulas  $\phi_1, \phi_2$ ,  $\phi^{imp}$  is the formula obtained as a boolean combination of  $\phi_1^{imp}$  and  $\phi_2^{imp}$ .

As we need to introduce imprecision through new variables, we replace every statement of the form  $\exists w_i, \phi$  by a statement of the form  $\exists w'_i, \exists x_i, \phi$ , and letting  $Q(V)||X$  denote the prefix obtained by replacement of every substring  $\exists w_i$ , by a string  $\exists w_i, \exists x_i$  in  $Q(V)$  and expression using We can now define the weakest precondition for  $\psi_{post}$  that has to be satisfied by  $D'_1, \dots, D_i, \dots, D'_k$  when enlarging field  $a_j$  as

$$wp[Enlargement_\delta]\psi_{post} = Q(V)||X, \phi_{post}^{imp}$$

One can notice that if  $\phi_{post}$  is separated (resp. in BSR fragment of FO), then  $wp[Enlargement_\delta]\psi_{post}$  is also separated (resp. in BSR fragment).  $\square$

LEMMA A.7 (WEAKEST PRECONDITION FOR UNIONS OF DATASETS). *Let  $\phi$  be a FO formula, and  $act$  be an atomic action that merges datasets with common relational schema. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  and  $P$  are separated FO formulas, then  $\psi$  is also separated.*

PROOF. **Union** operations merge datasets that have same relational schema. It takes data from different input datasets  $D_1, \dots, D_q$  with the same relational schema  $rs = (rn, A)$ , where  $A = (a_1, \dots, a_p)$  and produces an output dataset  $D_a$  with relational schema  $rs$ .

Let us assume that  $D'_1, \dots, D_a, \dots, D'_k \models \psi_{post}$ . We want to compute the weakest preconditions on  $D'_1, \dots, D_1, \dots, D_q, \dots, D'_k$ . As usual,  $\psi_{post}$  is an FO formula of the form  $Q(V), \phi_{post}$ . Now, every relation  $rn(w_i, w_{i+p}) \in D_a$  mentioned in the formula has to appear in a dataset  $D_i, i \in 1..q$ , that can be chosen when interpreting the formula. Similarly, if  $\psi_{post}$  contains a statement of the form  $\neg rn(w_i, w_{i+p}) \in D_a$ , then  $rn(w_i, w_{i+p})$  should not appear in any dataset  $D_i, i \in 1..q$ . Formula  $\psi_{post}$  holds for dataset  $D_a$  iff one can build a map  $aff : 1..KU \rightarrow 1..q$  that associates to every occurrence of relation  $rn$  a dataset from which a record instantiating relation  $rn(w_i, \dots, w_{i+p})$  originates. Let  $Assign(KU, q)$  denote the set of all possible assignments for the  $KU$  relations in  $\psi_{post}$ . For a particular assignment  $aff \in Assign(KU, q)$  we can write a formula  $\phi_{post}^{aff}$  where the  $m^{th}$  occurrence of  $rn(w_i, \dots, w_{i+p}) \in D_a$  is replaced by  $rn(w_i, \dots, w_{i+p}) \in D_{aff(m)}$ , and every occurrence of  $\neg rn(w_i, \dots, w_{i+p}) \in D_a$  is replaced by the conjunction  $\bigwedge \neg rn(w_i, \dots, w_{i+p}) \in D_i$ .

$D'_1, \dots, D_a, \dots, D'_k \models \psi_{post}$  iff one can find  $aff \in Assign(KU, k)$  such that  $D'_1, \dots, D_1, \dots, D_q, \dots, D'_k \models \phi_{post}^{aff}$ . Note that here, choices of records in different  $D_j$ 's are not independent (for some contents of input datasets, affecting  $rn(w_1, \dots, w_p)$  to  $D_1$  can impose to search a matching record for  $rn(w_{p+1}, w_{2.p})$  in another dataset).

Hence the weakest precondition on  $D'_1, \dots, D_1, \dots, D_q, \dots, D'_k$  such that  $\psi_{post}$  hold on  $D'_1, \dots, D_a, \dots, D'_k$  after merging  $D_1, \dots, D_q$  is

$$wp[Union]\psi_{post} = Q(V), \bigvee_{aff \in Assign(KU, k)} \psi_{post}^{aff}$$

As variables used in atoms of  $wp[Union]\psi_{post}$  do not change w.r.t the original formula, if  $\psi_{post}$  is separated, then all atoms in  $wp[Union]\psi_{post}$  also separate universally and existentially quantified variables.  $\square$

LEMMA A.8 (WEAKEST PRECONDITION FOR JOINS). *Let  $\phi$  be a FO formula, and  $act$  be an atomic action that performs a join between two datasets over a common field. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  and  $P$  are separated FO formulas, then  $\psi$  is also separated.*

PROOF. **Join** operations merge datasets with different relational schemas. For simplicity, we consider that joins apply to a pair of input datasets  $D_1, D_2$  with respective relational schemas  $rs_1 = (rn1, A_1)$  and  $rs_2 = (rn2, A_2)$  to

produce a output dataset  $D'_a$  with relational schema  $rs_a = (rn_a, B = A_1 \cup A_2)$ . We also assume that joins operate on equality of a single common field  $a_i$ . Without loss of generality, letting  $A_1 = (a_1, a_{p_1})$  and  $A_2 = (a'_1, \dots, a'_{p_2})$ , we consider that jointure on a common field is represented by  $a_{p_1}$  in  $rs_1$  and by  $a'_1$  in  $rs_2$ . That is, if a pair of records  $r_1 = rn_1(v_1, \dots, v_{p_1})$  and  $r_2 = rn_2(u_1, \dots, u_{p_2})$  have common value on their common field,  $v_n$  is the value of  $a_i$  in  $r_1$  and  $u_m$  the value of  $a_i$  in  $r_2$  then  $D_a$  will contain a record  $r = (v_1, \dots, v_p, u_2, \dots, u_{p_2})$ . Hence, letting  $D'_a, D'_3, \dots, D'_k$  be a set of datasets that satisfy a formula  $\psi_{post}$ , the weakest precondition that has to be computed is a property of  $D_1, D_2, D'_3, \dots, D'_k$ .

Formula  $\psi_{post}$  is an FO formula over a set of variables  $V = W \cup X$ , where  $W$  are variables involved in relational statements of the form in the form  $rn_a(w_i, \dots, w_i + p)$  or  $\neg rn_a(w_1, \dots, w_p)$ . Hence  $\psi_{post}$  is of the form  $Q(V), \phi_{post}$ . Now, every positive statement of the form  $rn_a(w_i, w_{i+p_1+p_2-1})$  mentioned in the formula originates from a pair of records in  $D_1, D_2$  with common value on  $a_i$ . Hence, every statement of the form  $rn_a(w_i, \dots, w_{i+p_1+p_2-1})$  holds for  $D_a$  iff the statements  $\phi_{EQ,i} = \exists x_i, rn_1(w_i, \dots, w_{i+p_1-1}) \in D_1 \wedge rn_2(x_i, w_{i+p_1} \dots, w_{i+p_1+p_2-1}) \in D_2 \wedge w_{i+p_1-1} = x_i$ , where  $x_i$  is a new variable that does not already appear in  $Vars(\psi_{post})$  holds. Similarly, for relational statement in negative form  $\neg rn_a(w_i, \dots, w_{i+p_1+p_2-1})$  holds for  $D_a$  iff the statement  $\bar{\phi}_{EQ,i} = \forall x_i, \neg(rn_1(w_i, \dots, w_{i+p_1-1}) \in D_1 \wedge rn_2(x_i, w_{i+p_1} \dots, w_{i+p_1+p_2-1}) \in D_2 \wedge w_{i+p_1-1} = x_i)$ , where  $x_i$  is a new variable that does not already appear in  $Vars(\psi_{post})$  holds. Let  $\psi_{post[D_a|D_1, D_2]}$  be the formula obtained by replacing every statement of the form  $rn_a(w_i \dots)$  by  $\phi_{EQ,i}$ , and every statement of the form  $\neg rn_a(w_i \dots)$  by  $\bar{\phi}_{EQ,i}$ . As  $x_i$ 's are fresh new variables, we can easily convert this formula into a prenex formula  $\psi_{post[D_a|D_1, D_2]}^{prenex}$  of the form  $Q(V). \exists x_1, x_k \forall x_{k+1}, \dots, x_{k+m} \phi$ , where  $x_1, \dots, x_k$  are fresh variables originating from positive relational statements and  $x_{k+1}, \dots, x_{k+m}$  originate from negative ones. Hence, the weakest precondition on  $D_1, D_2, D'_3, \dots$  for  $\psi_{post}$  to hold after a join is:

$$wp[Join]\psi_{post} = \psi_{post[D_a|D_1, D_2]}^{prenex}$$

One can immediately notice that if  $\psi_{post}$  is separated, then  $wp[Join]\psi_{post}$  is in separated form. As  $\psi_{post}$  is separated, all atoms either address properties of existentially quantified  $x_i$ 's or properties of universally quantified  $y_i$ 's. Hence, replacing a statement of the form  $rn(x_1, \dots, x_k)$  on existential variables with a statement of the form  $rn(x_1, \dots, x_q) \wedge rn(x_{q+1} \dots, x_k) \wedge x_1 = x_{q+1}$  in which all atoms address existentially quantified variables. For atoms with universally quantified variables, one can notice that  $\psi_{post}$  can be rewritten into an equivalent BSR formula. Hence, a formula of the form  $\exists \vec{Z}, \forall x, y, wrn(x, w, y)$  holds iff the precondition  $\exists \vec{Z}, \forall x, w, y, rn_1(x, w) \wedge rn_2(w, y)$ , which remains separated.  $\square$

LEMMA A.9 (WEAKEST PRECONDITION FOR RECORD INSERTION). *Let  $\phi$  be a FO formula, and act be an atomic*

*action that inserts a new record in a dataset. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  and  $P$  are separated FO formulas, then  $\psi$  is also separated.*

PROOF. **Record insertion** consists in adding a record to an existing dataset. Let  $D'_j$  be a dataset with relational schema  $rs = (rn, A)$  with  $A = (a_1, \dots, a_p)$ , and assume that  $D'_j$  is obtained after adding a record to an input dataset  $D_i$  with the same relational schema. Let  $D'_1, \dots, D'_j \dots D'_k \models \psi_{post}$ . When a set of records  $R$  selected from  $D'_i$ 's serves as a witness for the truth of  $\psi_{post}$  after an insertion, then at most one of these records or the form  $r = rn(v_1, \dots, v_p)$  can be the newly inserted tuple. That is,  $D'_1, \dots, D'_j \dots D'_k \models \psi_{post}$  iff  $D'_1, \dots, (D_i \uplus \{r\}) \dots D'_k \models \psi_{post}$ . It means that either  $D'_1, \dots, D_i \dots D'_k \models \psi_{post}$  or  $D'_1, \dots, D_i \dots D'_k \not\models \psi_{post} \wedge D'_1, \dots, D_i \uplus \{r\} \dots D'_k \models \psi_{post}$ . Let  $\psi_{post}$  be of the form  $Q(V), \phi$ , with  $V = W \cup Y$ , and  $W$  be the variables appearing in relational clauses of the form  $rn(w_i, \dots, w_{i+p})$ . Let us assume that  $\phi$  is a quantifier free formula in disjunctive normal form. In other words,  $\phi$  is of the form  $\phi = \bigvee_{k=1..K} \phi_k = at_{k,1}(V) \wedge \dots \wedge at_{k,m_k}(V)$ , where each  $at_{k,k'}(V)$  is an atom involving a subset of variables in  $V$ . If  $Q(V), \phi_k$  is separated, then one can compute an equivalent formula in BSR form of the form  $\exists \vec{V}_1, \forall \vec{V}_2 \phi'_k$ . This formula is satisfied if one can find an assignment of variables in  $V_1$  such that for every assignment of variables in  $V_2$ ,  $\phi'_k$  evaluates to true when replacing variables by their value. All existential variables are separated. For existential variables under the scope of a relational statement  $rn(w_i, \dots, w_{i+p})$ . Let  $AK$  be the number of relational statements of the form  $rn(\dots) \in D'_j$ . One can hence choose, for each statement  $rn(w_i, \dots, w_{i+p})$  whether variables  $w_i, \dots, w_{i+p}$  are assigned values freshly introduced by the newly created record or not. In the first case, one can relax constraints on  $w_i, \dots, w_{i+p}$  in the precondition, i.e., remove all atoms of the form  $rn(w_i, \dots, w_{i+p})$  or  $P(X)$  where  $X \subseteq \{w_i, \dots, w_{i+p}\}$  and eliminate the variables from arithmetic predicates: for a predicate  $P(w_i, \dots, w_{i+p}, x, y, z, \dots)$  that imposes linear constraints on the values of variables, one can use elimination techniques such as Fourier-Motzkin to compute a new predicate  $P'$  on  $x, y, z, \dots$ . We hence have  $2^{AK}$  possible assignments. For every possible assignment  $Ass_X$ , we can define the set  $V_X$  of variables that can be eliminated, and we can compute the formula  $\phi_{k \setminus Ass_X}$  that eliminates relational statements matching the newly inserted record according to  $Ass_X$  from  $\phi_k$ , and computes new predicates. Hence, under the assumption that  $Ass_X$  is a correct assignment,  $D'_1, \dots, D'_j \dots D'_k \models \phi_k$  iff  $D'_1, \dots, D_i \dots D'_k \models \phi_{k \setminus Ass_X}$ . For  $\psi_{post}$  to hold after insertion, there must be at least a correct assignment.

The precondition for  $\psi_{post}$  that has to be satisfied by  $D'_1, \dots, D_i \dots D'_k$  hence becomes:  $wp[Additive]\psi_{post} = \bigvee_{k=1..K} \exists V_1 \setminus V_X \forall V_2 \phi_{k \setminus Ass_X}$ . One can notice that if  $\psi_{post}$  is separated, the resulting formula is still separated.  $\square$

LEMMA A.10 (WEAKEST PRECONDITION FOR RECORD DELETION). *Let  $\phi$  be a FO formula, and  $act$  be an atomic action that removes a record from a dataset. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  is a separated FO formulas, then  $\psi$  is also separated.*

PROOF. **Record deletion** removes a record from an existing dataset. Let  $D_j$  be a dataset with relational schema  $rs = (rn, A)$  with  $A = (a_1, \dots, a_p)$  such that  $D'_1, \dots, D'_j \dots D'_k \models \psi_{post}$  and is obtained after deletion a record from an input dataset  $D_i$  with the same relational schema. Let  $r = rn(v_1, \dots, v_p)$  be the tuple removed from  $D_i$ . We can rewrite the statement  $D_j \models \psi_{post}$  as  $D_i \setminus \{r\} \models \psi_{post}$ . Now, for every possible instance of record  $r$  there are two possibilities: either presence of  $r$  does not falsify  $\psi_{post}$ , or  $r$  is a record that falsifies  $\psi_{post}$  if it appears in dataset  $D_i$ .

In the first case, we have that  $D_i \models \psi$ . In the second case, we have that  $D'_j \uplus \{r\} \models \neg\psi$ . Formula  $\neg\psi$  is a separated formula obtained in the usual way by inverting existential and universal quantifiers in the prefix of  $\psi$  and negation of atoms in the matrix. As existence of record  $r$  is required we have that  $r$  necessarily matches (at least) one of the positive relational statements  $rn(w_i, ..w_{i+p})$  in  $\neg\psi$ . As for record additions, we can build a formula  $\neg\psi_X$  in BSR form that should hold under the assumption that assignment  $X$  assigns the field values of  $r$  to some relational statements of  $\neg\psi$ . More precisely,  $\neg\psi_X$  is the formula obtained by removing relational statements assigned to  $r$  in the BSR form computed from  $\neg\psi$ . The weakest precondition for deletion that has to be satisfied by  $D'_1, \dots, D_i \dots D'_k$  becomes :  $wp[Del]\psi = Q(V), \psi \wedge \bigwedge_{X \in Ass} Q(V) \neg\psi_X$ . Notice that if  $\psi$  is in separated form, then  $wp[Del]\psi$  is also in separated form.  $\square$

LEMMA A.11 (WEAKEST PRECONDITION FOR DATASETS DECOMPOSITION). *Let  $\phi$  be a FO formula, and  $act$  be an atomic action that decomposes a dataset into smaller datasets. Then one can effectively compute an FO formula  $\psi = wp[act]\phi$ . Moreover, if  $\phi$  and  $P$  are separated FO formulas, then  $\psi$  is also separated.*

PROOF. The **Decomposition** of a task is a higher order operation to split a task into several orchestrated sub-tasks. In particular, this operation splits an input dataset  $D_i$  with relational schema  $rs = (rn, A)$  into a set of datasets  $D'_1, \dots, D'_l$ . Each  $D'_j, j \in 1..l$  is obtained through application of a function  $f_j$  to the input dataset  $D_i$ . The relational schemas  $rs_j = (rn_j, A_j)$  of  $D'_j$ 's need not be the same as  $rs$ . Property  $\psi$  is of the form  $\psi = Q(V), \phi$ , and such that  $\phi$  contains positive relational statement of the form  $rn_j(w_1..w_{|A_j|}) \in D'_j$ , and negative relational statements of the form  $\neg rn_j(w_1..w_{|A_j|}) \in D'_j$ . As we have that  $D'_j$  is a dataset obtained as a function  $f_j(D_i)$  we can rewrite  $\psi$  as an equivalent formula  $\psi_2 = Q(V), \phi_2$ , where  $\phi_2$  is obtained by replacing every instance of  $rn_j(w_i..w_{i+|A_j|-1}) \in D'_j$  by  $rn_j(w_i..w_{i+|A_j|-1}) \in f_j(D_i)$  in formula  $\phi$ . Let us assume that  $f_1, \dots, f_l$  are simply selections of records according to predicates  $P_1, \dots, P_l$  that form a partition of  $D$ . Let  $p = |A|$ .

Statement  $rn(w_i, w_{i+p}) \in f_j(D_i)$  holds iff there exists a record  $r = (v_1, \dots, v_p)$  in  $D_i$  that is a solution for formula  $P_j(w_i, \dots, w_{i+p})$ . Equivalently, a positive statement of the form  $rn_j(w_k, \dots, w_{k+|A_j|-1}) \in f_j(D_i)$  can be replaced by  $rn_j(v_k, \dots, v_{k+|A_j|-1}) \in D_i \wedge P_j(w_k, \dots, w_{k+|A_j|-1})$ , and a negative statement of the form  $\neg rn_j(v_k, \dots, v_{k+|A_j|-1}) \in f_j(D_i)$  can be replaced by  $\neg (rn_j(v_k, \dots, v_{k+|A_j|-1}) \in D_i \wedge P_j(w_k, \dots, w_{k+|A_j|-1}))$ . Letting  $\phi_3$  be the formula  $\phi_2$  where every relational statement has been replaced this way, and letting  $Q'(V)$  denote the prefix in which every instance of  $w_k, \dots, w_{k+|A_j|-1}$  is replaced by fresh variables  $v_k, \dots, v_{k+|A_j|-1}$ , the weakest precondition needed such that  $D'_1, \dots, D_l \dots D'_k \models \psi$  is hence

$$wp[Decomp]\psi = Q'(V), \phi_3$$

If one function  $f_j$  is not simply a selection but also computes new attributes for records in  $D_j$  from values attached to variables  $v_k, \dots, v_{k+|A_j|-1}$  then one can also replace  $rn_j(w_k, \dots, w_{k+|A_j|-1})$  by another FO formula following the lines of R2R replacement. We leave details of the construction to readers.

Let us illustrate it with a small example. Let  $D'_1, D'_2 \models \exists x, y, z, t, rn_1(x, y) \in D'_1 \wedge rn_2(z, t) \in D'_2 \wedge y = z$ , with  $rs_1 = (rn_1, \{a_1, a_2\})$   $rs_2 = (rn_2, \{a_3, a_4\})$ , and  $Dom(a_1) = dom(a_2) = dom(a_3) = dom(a_4) = \mathbb{R}$ . Let us assume that  $D'_1$  and  $D'_2$  are obtained by decomposition of an input dataset  $D_i$  with relational schema  $rs_i = (rn_i, \{b_1, b_2\})$ , through selection with selection predicates  $P_1 ::= b_1 < 10$  and  $P_2 ::= b_1 < b_2$ . Then,  $wp[Decomp]\psi$  is the formula

$$D_i \models \exists v_1, v_2, z, t, \quad rn_1(v_1, v_2) \in D_i \wedge v_1 < 10 \\ \wedge rn_2(v_3, v_4) \in D_i \wedge v_3 < v_4 \wedge v_2 = v_3$$

Data distribution performed by splits is mainly a generalization of selection. Indeed if  $\psi_{post}$  is a separated formula, then all atoms in  $wp[Decomp]\psi$  are separated, and  $wp[Decomp]\psi$  a separated formula.  $\square$

Now that we have defined weakest preconditions for basic operation that manipulate data, we can formalize how these conditions are associated to steps along a run of a complex workflow. Let  $\rho = C_0 \dots C_n$  be a run, that ends in a configuration where a node  $n_k$  with input data  $D_k$  can be split. We will define inductively a sequence  $WP_0 \dots WP_{n-1}$  of conditions to be met at each stage such that condition  $D_n = \emptyset$  is met at step  $n$  (hence leading to an unavoidable deadlock). If a condition  $WP_i = D'_1, \dots, D'_m \models \psi$  has to be met when reaching a configuration  $C_i$ , then the condition associated with  $WP_{i-1}$  is the weakest precondition such that  $WP_i$  holds. Depending on the nature of the move  $C_{i-1} \rightarrow C_i$ ,  $WP_{i-1}$  is of the form  $D'_1, \dots, D'_q \models \psi_{i-1}$ , where  $\psi_{i-1}$  is computed inductively as  $wp[op_1](wp[op_2](\dots wp[op_k]\psi_i))$ , and  $op_1, \dots, op_k$  is the sequence of operations used to transform datasets  $D_1, D_q$  in  $C_{i-1}$  into  $D'_1, \dots, D'_m$  in  $C_i$ . The weakest precondition for move from  $C_{i-1}$  to  $C_i$  is hence  $D'_1, \dots, D'_m \models wp[op_1](wp[op_2](\dots wp[op_k]\psi_i))$ .

For automatic actions executions, the operation used is a combination of selection, projection, R2R transformations and the weakest precondition follows the rules defined above.

For splitting, the operation used is a decomposition of a particular dataset according to a set of functions  $f_1, \dots, f_k$ , to obtain new datasets and new data assignments. If  $WP_i$  is of the form  $D_1, \dots, D_m$  and datasets  $D_n, \dots, D_{n+k}$  are obtained by splitting a node and its input data  $D$  then  $WP_{i-1}$  is of the form  $D_1, D_{k-1}, D, D_{n+k+1} \models wp[decomp]\psi_i$ . For user actions that are input or deletions, one transforms a single datasets  $D_j$  and  $WP_{i-1}$  is of the form  $D_1, \dots, D'_j, \dots, D_m \models \wp[add/remove]\psi_i$ . Last, moves that simply perform user assignments do not change the nature of conditions that have to be met by a set of datasets. Let  $D_1, D_m \models \psi$  be the condition that has to be met at step  $k$  of a run  $\rho$  and let the move from  $C_{k-1}$  be an user assignment. Computing the weakest preconditions for addition of records calls for the use of an elimination step. Let  $\psi$  be an FO formula, with equality only. This formula can encode properties of the form  $x + 1 < y$  as boolean relations of the form *Plus1 - LessThan*( $x, y$ ). More generally, an inequality of the form  $x + k < y$  can be encoded as a boolean statement *Plusk - Lessthan*( $x, y$ ). Conversely, one can syntactically transform every expression of the form *Plusk - Lessthan*( $x, y$ ) into an inequality  $x + k < y$ . Now, when eliminating a variable  $y$  from a system of inequalities with equations of the form  $x + k < y$  and  $y + k' < z$  one may obtain an inequality of the form  $x + (k + k') < z$ . That is, if one converts again this inequality into a boolean assertion, one needs to use one more binary predicate. Hence, at every weakest precondition computation, the number of side arithmetic predicates in use increases. This is not a problem in our case however, as a finite number of new predicates is produced at each step, and the number of weakest precondition to compute is also bounded.

LEMMA A.12. *Let  $\rho = C_0 \dots C_n$  be a path of the execution tree, where  $C_n$  is a configuration that allows for the split of a particular dataset  $D_n$ . Let  $W_n ::= D_n = \emptyset$ , and  $W_0, \dots, W_{n-1}$  be the weakest preconditions computed for each step of  $\rho$ . Then, the number of side arithmetic predicates used to define  $WP_0, \dots, WP_n$  is bounded.*

PROOF. The elimination of variables while deriving the weakest precondition is carried using Fourier-Motzkin elimination technique (see appendix B.2). During each step, running an elimination step of one variable over  $m$  number of linear inequalities results into at most  $m^2/4 = \theta(m^2)$  linear inequalities in worst case. If we remove  $k$  number of variables, the algorithm must perform  $k$  step, hence the worst case the algorithm takes is  $\theta(m^{2k})$ . FME may result into redundant set of linear inequalities. The detection and elimination of redundant variables is trivial and can be done using principle of linear programming. The scope of removal of redundant linear inequalities is beyond the scope of this paper. Now, in context to our problem, the total number of weakest precondition that needs to be calculated is  $n$ . Let  $m_i$  be number of linear inequalities and  $k_i$  denote the number of variables that need to be eliminated for the derivation of each  $w_i$  in  $\rho$ . Hence, in the worst case the total number of new linear inequalities becomes  $\sum_{i=1}^n m_i^{2k_i}$ . Now these inequalities can

be expressed in terms of boolean assertion to get the predicates. Henceforth, as the number of new linear inequalities is bounded, we infer that the number of side predicate is also bounded.  $\square$

LEMMA A.13. *Let  $\rho = C_0 \dots C_n$  be a path of the execution tree, where  $C_n$  is a configuration that allows for the split of a particular dataset  $D_N$ . Let  $WP_n ::= D_n = \emptyset$ , and  $WP_0, \dots, WP_{n-1}$  be the weakest preconditions computed backwards for each step of  $\rho$ . Then, every  $WP_j, j \in 1..n-1$  is a weakest precondition of form  $WP_j = D_1, \dots, D_{m_j} \models \psi_j$  where  $\psi_j$  is a separated FO formula.*

PROOF. The proof follows from the lemma 2. In a run  $\rho$ , every move  $m_i$  transform a set of input data to output data using the transform function  $f_i$ . Let  $v_n$  be the split node in the execution tree resembling the configure  $C_n$ . We compute the weakest precondition on the backward path from the node  $v_n$  to the root node  $v_0$  as  $v_n \rightarrow v_{n-1} \rightarrow v_0$ . At each node  $v_j$  of the execution, there exist a function  $f_j$  which transform the input dataset  $D_1, \dots, D_{m_j}$  to the corresponding set of output dataset. As per lemma 2, for every move  $m_i$ , we can compute an effective weakest precondition  $wp[m_i]\psi_{post_i}$ . The weakest precondition is a FO formula  $\psi_j$  that holds on input dataset  $D_1, \dots, D_{m_j}$  such that after execution of the function  $f_j$ , the output dataset must satisfy the given post condition  $\psi_{post_j}$ . Hence, for every move  $m_j$  there exist a weakest precondition  $WP_j$  such that  $D_1, \dots, D_{m_j} \models \psi_j$ .  $\square$

With all the above lemmas, we have shown that the weakest precondition for an FO formula and actions that are projections, deletion or insertion of records, field addition, splits of datasets, joins, atomic execution of tasks transforming one record or all records, application of linear transformation of records. All actions occurring during the execution of a complex workflows can be expressed as a sequence of all these basic transformation of datasets (for instance, insertion of imprecise data can be seen as an insertion of a record followed by a linear transformation. As all weakest preconditions for basic transforms of dataset are FO formulas, and as separated formulas also give separated weakest preconditions, we obtain our result.  $\square$

## B ADDITIONAL MATERIAL

For the convenience of readers, this section provides additional material on Symbolic execution trees and on the elimination technique (Fourier-Motzkin) used to compute new predicates on record.

### B.1 Symbolic Execution Tree

Remind that a symbolic execution tree is a tree  $(V, E)$  where every vertex represents a set of configurations of a complex workflow that only differ w.r.t. their data assignment, and  $E$  represents moves among these configurations (user assignments, task executions, refinements). Figure 3 represents a symbolic execution tree. Vertices of the tree are represented by circles. Deadlocked vertices are represented by dotted

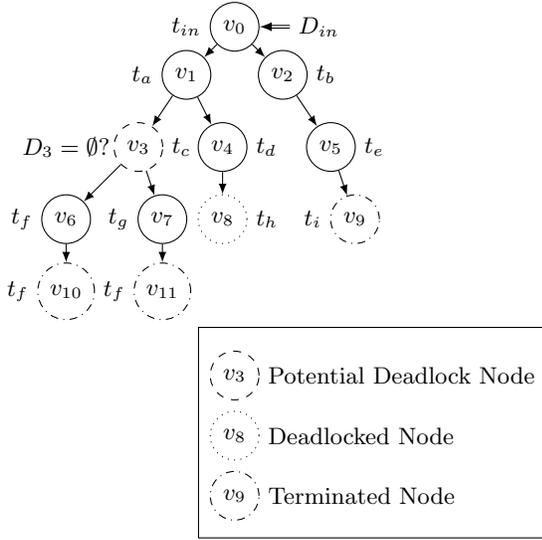


Figure 3: Symbolic Execution Tree

circles, terminated vertices by dotted-dashed circles, and potential deadlocks by circles with dashed lines. Two vertices  $v_i, v_j$  are connected by an arrow iff there exists an action (user assignment, task execution, complex task refinement) that transforms the configuration represented by vertex  $v_i$  into another configuration represented by vertex  $v_j$ .

If the execution tree of a complex workflow contains a deadlocked vertex, then obviously all executions of the workflow do not terminate, as there is a path from the initial configuration to a deadlocked situation, and that the sequence of actions represented by this path cannot be prevented by the contents of data forged during the execution.

If the execution tree contains a potential deadlock  $V_i = (W_i, Ass_i, \mathbf{Dass}_i^S)$ , then the workflow part  $W_i$  of this vertex contains a split node  $n$ , minimal in the workflow. To be able to split and distribute data,  $\mathbf{Dass}_i(n)$ , the data input to  $n$  should not contain an empty dataset. Otherwise, an execution starting from a configuration of the form  $C_i = (W_i, Ass_i, \mathbf{Dass}_i)$  will eventually deadlock. On the Figure, the property to check is that no execution ending in a configuration with signature  $v_3$  and such that dataset  $D_3$  is empty is accessible. In this example, if there is no way to derive preconditions for  $D_{in}$  such that  $D_3 = \emptyset$ , then the split operation can be done safely, and all executions starting from  $v_3$  terminate.

## B.2 Elimination with Fourier-Motzkin

The Fourier-Motzkin Elimination (FME) technique is a standard algorithm to eliminate variables and solve systems of linear inequalities. Let  $X = \{x_1, \dots, x_k\}$  be a set of variables. A system of linear inequalities over  $X$  is an expression of  $\gamma ::= A.X \leq B$ , i.e. a collection of inequalities of the form  $a_1.x_1 + a_2.x_2 + \dots + a_k.x_k \leq b$ . Given a variable  $x_i$ , the FME technique computes a new system of inequalities

$\gamma' ::= A'.X' \leq B'$  over  $X' = X \setminus \{x_i\}$ , and such that  $\gamma$  has a solution if and only if  $\gamma'$  has a solution. The algorithm works in three steps:

**Step 1:** Normalize all inequalities in  $\gamma$ , i.e. rewrite every inequality containing  $x_i$  of the form

$$a_1.x_1 + a_2.x_2 + \dots + a_i.x_i + \dots + a_k.x_k \leq b$$

into a new inequality of the form

$$x_i \leq \frac{b}{a_i} - \frac{a_1}{a_i}.x_1 - \frac{a_2}{a_i}.x_2 - \dots - \frac{a_k}{a_i}.x_k$$

or

$$x_i \geq \frac{b}{a_i} - \frac{a_1}{a_i}.x_1 - \frac{a_2}{a_i}.x_2 - \dots - \frac{a_k}{a_i}.x_k$$

**Step 2:** separate the obtained system into  $\gamma^+, \gamma^-, \gamma^0$ , where  $\gamma^+$  contains all inequalities of the form

$$x_i \geq f(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k),$$

$\gamma^-$  contains all inequalities of the form

$$x_i \leq f(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k),$$

and  $\gamma^0$  all other inequalities that do not refer to  $x_i$ .

**Step 3:** create a new system of inequalities that contains  $\gamma^0$  and, for each pair of inequalities

$$x_i \leq f_1(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k) \in \gamma^-$$

and

$$x_i \geq f_2(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k) \in \gamma^+,$$

a fresh inequality of the form

$$f_2(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k) \leq f_1(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k)$$

The new system obtained is still a system of linear inequalities. It does not contain variable  $x_i$  and is equivalent to the original system.