Data Centric Workflows for Crowdsourcing

Pierre Bourhis, Loïc Hélouët, Rituraj Singh, Zoltán Miklós

Data Centric Workflows for Crowdsourcing

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

[1] CNRS, Univ. Lille
pierre.bourhis@univ-lille1.fr
[2] INRIA
loic.helouet@inria.fr
[3] Univ. Rennes
rituraj.singh@irisa.fr,zoltan.miklos@univ-rennes1.fr

Abstract. Crowdsourcing is a major paradigm to accomplish works that require human skills, by paying a small sum of money and drawing workers all across the globe. However, crowdsourcing platforms are mainly ways to solve large amounts of relatively simple and independent replicated work units. A natural extension of crowdsourcing is to enhance the definition of work, and solve more intricate problems, via orchestrations of tasks, and via higher-order, i.e. allowing workers to suggest *a process to obtain data* rather than a returning a *plain answer*. This work proposes *complex workflows*, a data centric workflow model for crowdsourcing. The model 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 consider termination questions: We show that existential termination (existence of at least one terminating run) is undecidable excepted for specifications with bounded recursion. On the other hand, universal termination (whether *all* runs of a complex workflow terminate) is decidable (and at least in $co-2EXPTIME$) when constraints on inputs are specified in a decidable fragment of FO. We then address correctness problems. We use FO formulas to specify dependencies between input and output data of a complex workflow. If dependencies are specified with a decidable 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 with some restrictions.

1 Introduction

Crowdsourcing is a powerful tool to leverage intelligence of crowd to realize tasks where human skills still outperform machines [18]. It has been successful in contributive science initiatives, such as CRUK's Trailblazer[4], Galaxy Zoo[5], etc. Most often, a crowdsourcing project consists in deploying a 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 participation 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. Indeed, many projects, and in particular scientific workflows, take the form of orchestrations of high-level composite tasks. Each high-level task can be seen individually as a data collection task, or as a processing of a large dataset, built as the union of results from independent easy micro-tasks. However, the coordination of these high-level tasks to achieve a final objective calls for more evolved processes. One can easily meet situations in which the result of a high-level task serves as entry for the next stage of the overall process: for instance, one may want to remove from a picture dataset images of poor quality before asking users to annotate them. Similarly, there are situations allowing parallel processings of datasets followed by a merge of the obtained results. A typical example is cross-validation of answers returned by different users.

[4] https://www.cancerresearchuk.org
[5] http://zoo1.galaxyzoo.org

As noted by [31], composite tasks are not or poorly supported by crowdsourcing platforms. Crowdsourcing markets such as Amazon Mechanical Turk[6] (AMT), Foule Factory[7], CrowdFlower[8], 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. The workflow of such processes is hence dynamic and shall consider users skills, availability, the data they produce, but also their knowledge about processes themselves. 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. Thus 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 allow higher-order answers to refine processes dynamically static orchestration of easy micro-tasks before execution of this low-level workflow. However, this solution lacks adaptability, and may miss some interesting skills of participants who cannot realize directly 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 implementation of complex tasks. In addition, 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.

Now, it is frequently admitted that the main difficulty to use crowdsourcing platforms is the lack of solutions to build these high level orchestrations. Running a complex process consists in, roughly speaking, refining a high-level task into a composition of finer grain subtasks with a divide and conquer strategy. Orchestration languages such as ORC [15, 25], BPEL [3, 28], etc. already exist, and could easily serve as a basis for coordination of high-level tasks. However, in data-centric models involving human stake holders, skills, time, etc, one expects solutions to adapt a process at runtime depending on the data contents, on availability of workers, etc. Models such as ORC, BPEL, and many process algebras they are tailored for workflows which dynamics is fixed a priori and cannot evolve at runtime and do not fulfill this additional requirement. Some solutions for the design and monitoring of complex processes running on crowdsourcing platforms have been proposed. [29] propose a timed modeling language for complex tasks based on adaptive workflow net. A graphical net with deadline mechanism is presented to design, describe and visualize the flow of tasks at crowdsourcing platforms. Kulkarni et. al propose the Turkomatic tool which works on the principle of Price, Divide and Solve (PDS) [19]. The tool uses the power of crowd to decompose a complex task while the requester can monitor the decomposition workflow. Crowdforge is an interesting work which uses Map-Reduce technique to solve complex tasks [16]. It provides a graphical web interface and primitives (partition, map and reduce) to decompose complex tasks into sub-tasks and define dependencies among them. Zheng et. al [33] define a PDS framework based on refinement of state machines.

Providing tools to implement PDS solutions is already a progress for crowdsourcing. We advocate, however, that an implicit demand of a requester is that the orchestration of micro-tasks that will be performed to realize his complex process will terminate, and will return appropriate answers. Further, we propose to rely on human skills to propose at runtime correct and efficient decomposition of tasks. In this work, we define a workflow language where the relation between data input to a task and output by it is formally defined, and in which users can propose refinements of a task, in addition to their standard contribution to

[6] https://www.mturk.com
[7] https://www.foulefactory.com
[8] https://www.crowdflower.com

micro-tasks. These refinements depict *the way to obtain an answer* rather than the answer itself. We then address the question of termination and correctness of complex workflows defined with this formalism.

In this paper, we focus on the orchestration of complex tasks in a crowdsourcing environment, with higher order constructs allowing online decomposition of the tasks by crowdworkers. A complex task is defined as a workflow orchestrating sub-tasks. 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 called *complex workflows*, allowing for the definition of data-centric workflows with higher-order schemes allowing workers 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 and 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 the 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 $workflows$, mainly at the level of micro-tasks [11, 21]. Crowdforge uses Map-Reduce techniques along with a graphical interface to solve complex tasks [16]. Turkit [22] 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 [19] 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 crowd workers 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 [33] 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. The 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. We do not claim exhaustiveness, and mainly refer in the rest of this section to a few papers with models or objectives seems the closest to our model and verification solution. Workflow nets [32] 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. As already mentioned, orchestration models originally built for business processes such as ORC [15, 25] or BPEL [3, 28] are not a priori tailored for dynamic orchestrations nor for handling datasets manipulations (they are more centered on the notion of transaction than on the notion of datasets transformations). Data-centric models and their correctness have also been considered in many papers. 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 (LTLT-FO with past operators) is de-

cidable. Business artifacts were originally developed by IBM [27], and verification mechanisms for LTL-FO were proposed in [7, 17] for subclasses of artifacts with data dependencies and arithmetic. LTL-FO formulas are of the form $\forall x_1, \ldots, x_k, \phi$ where ϕ is an LTL formula including FO statements. Variables are always universally quantified. [9] 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 tasks refinement. Further, LTL-FO verification focuses mainly on dynamics of systems (termination, reachability) but does not address correctness. [12] considers Data-centric dynamic systems (DCDS), i.e. relational databases equipped with guarded actions that can modify their contents, and call external services. This work proposes verification techniques for fragments of First-Order μ-calculus, and show decidability for run-bounded systems, in which external services cannot be repeatedly called with arguments returned by former service calls. Run boundedness is undecidable, but weakly-acyclic DCDS enjoy this property.

The model proposed in our paper is a workflow with higher-order constructs that refine tasks by new workflows, dependencies between data input and output by a task, and in which causal flows among tasks indicate how data in passed from one stage of the computation to the other. [4] proposes a verification scheme for a similar model depicting business processes with external calls. The business processes are recursive functions that can call one another, and the verification that is performed is defined using a query language. This language specifies shapes of workflow executions with a labeled graph and transitive relations among its nodes. A query Q is satisfied by a workflow W if there is an embedding of Q into the unfolding of W. While this model addresses the shape of executions, it is more focused on the operational semantics of workflows. A contrario, our verification scheme does not address the shape of control flow, but rather the contents of data manipulated by the workflow, i.e. it is more concerned by the denotational semantics of the specification, and on the question of termination.

This paper is organized as follows: Section 3 introduces our model. Section 4 defines its operational semantics, and section 6 addresses the termination question. Section 7 considers proper termination of complex workflows, before conclusion. For readability reasons, some technical proofs are only sketched, but their full versions are provided in appendix.

2 Motivation

Our objective is to provide tools to develop applications in which human actors are involved to resolve tasks or propose solutions to complete a complex task. The envisioned scenario is the following: a client provides a coarse grain workflow depicting important phases of a complex task to process data, and a description of the expected output. The tasks can be completed in several ways, but cannot be fully automated. It is up to a pool of crowdworkers to complete them, or to refine the tasks up to the fine grain level where high-level tasks are expressed as orchestrations of basic simple tasks.

2.1 A simple example: the actor popularity poll

To illustrate the needs for complex workflows, refinement, and human interactions, we give a simple 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. This ranking is a sent to a crowdsourcing platform as a high-level process decomposed into three sequential phases: first a collection of the most popular actors, then a selection of the 50 most cited names, followed by a classification of these actors in comedy/drama/action category. The ranking ends with a vote for each category, that asks contributors to associate a score to each name. The client does not input data to the system, but has some requirements on the output: 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. Form 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. This simple sequential task orchestration is depicted in Figure 1. The workflow starts from and empty input dataset $D_\emptyset$, and should return the desired actor popularity data in a dataset D_{ranked}.

The model proposed in section 3 is tailored to realize this type of application with the help of workers registers on a platform who can either provide their experience on how to decompose difficult tasks, or input their knowledge and opinion on the platform.

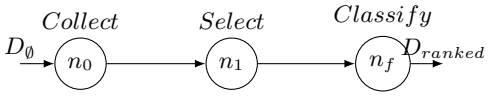

Fig. 1. A simple actor popularity poll.

2.2 A real field example: the SPIPOLL initiative

We provide a real field example, where managing task decomposition, orchestration, and workers contributions are key issues. We study the SPIPOLL initiative [9], a participatory science project. The project aims at collecting quantitative data on pollination, flowering insects to measure the diversity and network structure of pollination in France. This task is not trivial for several reasons. First of all, collecting information on insects requires a huge workpower to collect significant samples throughout the whole country. Second, the project lacks workpower to sort, classify, huge datasets , and then derive conclusions from the observed populations. While volunteers can help sorting pictures of insects, some rare taxons can only be recognized by experts. The objective of SPIPOLL is to organize a collaborative work involving ordinary people, trained volunteers and experts in order to build significant datasets and extract information on pollinating insects populations. We believe that this project can benefit from advances in data centric crowdsourcing platforms, and in particular for the design and automation of complex data acquisition procedures.

Roughly speaking, the high-level description of SPIPOL tasks is: acquire data (pictures), classify them, and publish the results. The result of classification should be a set of images with location and time tags, together with a taxon chosen among a finite set of species. The data acquisition and classification protocols specified by SPIPOLL fits well with the core idea of complex tasks. The protocol proposed by SPIPOLL is following. Volunteers stand for a certain duration (usually 20 minutes) in a place frequented by insects (bushes, flower beds,...) and take pictures of insects pollinating on a flower in this place. Once this phase is completed, the observer uploads his pictures on a server. This first phase results in huge collections of pictures, but not all of them are exploitable. The obtained result after the first phase is a dataset which records are of the form of $R = \{image, place, time\}$. Data collected from various sources are used as inputs for a second phase. In this second phase, workers have to classify pictures according to their quality, i.e. identify images that will help finding a correct identification for the represented insect. As the images are collected from diverse and unknown observers, there is need to rank and tag the respective image based on their quality, i.e. associate with each image a tag $quality = \{poor, average, good, best\}$. Generally, for these tasks, considering the bias among the workers, a typical image is given to k different workers. The workers give their opinion and tag each of the obtained images.

The third phase aggregates the result obtained in the second phase and gives a final verdict for each picture. Majority voting technique are widely used to reach the common consensus among the workers.

[9] http://www.spipoll.org/

The fourth phase removes images with *poor* quality from the dataset. The fifth phase, categorizes the insect images with sufficient quality obtained as input from the fourth phase into different species category, $category = \{category_1, category_2, \ldots, category_l\}$. Similar to the second phase, an image is distributed to n unique workers who are asked to tag a *category* to the insect image. The output of this phase serves as input for a sixth phase that aggregates the result by majority voting to obtain the $final\ category$ for each picture, and then passes the result for the final assessment. The final assessment for each image is performed by the experts or scientists for validation and then the obtained results are published. These types of complex tasks can be often found when there is need of data collection, data cleaning and then processing the data based on human intelligence.

The SPIPOLL example raises several observations on the realization of complex tasks in a crowsourcing environment. First, one can note that some of the tasks need to be completed sequentially, and on the other hand some tasks can be executed in parallel. For example, in the *Spipoll* case study, images can be only be annotated after they have been collected by observers. As a result, the first phase and the second phase can only be executed sequentially. However, images can be annotated in parallel (second, fifth phase) by different sets of workers. Another remark is that some recurrent orchestration patterns appear, such as dataset distribution, tagging and aggregation. These work distribution patterns are typical examples where crowdsourcing platforms are particularly adapted to improve quality and delays in data analysis. The dataset obtained from the observers can be large and annotating such big datasets is usually a task that cannot be performed by a single worker. Most often, volunteer contributors in crowdsourcing platforms spend little amounts of time on the platform. Hence the annotation phase needs to be decomposed, for instance by splitting large datasets into several chunks of reasonable size. These small chunks of data are then distributed and tagged in parallel by several workers, before aggregation. In this setting, refinement of a heavy tasks into several smaller concurrent easy tasks saves a lot of time.

Another lesson learned from the SPIPOLL case study is that some tasks may require special skills, such as expertise on insects to be completed. The validation task in the above example requires specialized people to judge the validity of proposed taxons. Hence, the system should consider the expertise of the worker in allocating tasks. Another thing to note that the example depicted here is data-centric, i.e. the role of each high-level or low-level task is to manipulate and transform data (select, tag,...) One can also observe that all tasks do not require human intelligence, and some of them can be easily performed by machines. As for example, consensus between the workers are performed by applying majority voting techniques. Other tasks such as selection of best pictures are roughly speaking SQL queries and can be automated. Thus, the system is a mixture of human and machine powered intelligence. Generally, machine powered tasks are deterministic in nature and human doable tasks comes with uncertainty.

As already mentioned, the output of one task can be needed as input to another task, which leads to causal dependencies among tasks. In addition, the execution of one task may affect the output of another task. An erroneous output of one task can jeopardize the execution of several successive tasks and in the long run may also halt the whole process. Hence, there needs verification techniques to ensure consistency and guaranteed output for each of the task execution.

2.3 Problems for Complex Workflows

Many existing crowdsourcing applications collect the result of hundreds of micro-tasks, and then aggregate the obtained answers to produce an output. In this simple setting, there is no doubt on the termination of the application : once enough data is collected, the aggregation phase can most of the time be reduced to an automated statistical analysis of returned answers. If incentives and difficulty are properly set, the data collection will terminate in a reasonable amount of time.

In more complex processes requiring refinement of tasks and calling for particular competences, offering the capacity to refine complex tasks may cause undesired behaviors in a workflow: workers can introduce deadlocks, unbounded recursion,... Even with very limited refinement capacities, it is hence not guaranteed that an application always terminates, if it terminates when appropriate data is used as input, etc. When a complex workflow terminates, another question is whether the data forged during the execution of the workflow satisfies the requirements set by the client of the application (in the case of the actor poll, for

instance, every listed actor must have a number of cites greater than 0). In the rest of the paper, we formalize complex workflows, and address termination questions. More precisely, given a complex workflow and a set of workers, we will address the following problems:

- **(universal termination)** Does the workflow terminates for every possible input to the system ?
- **(existential termination)** Is there at least one input for which at least one execution of the workflow terminates ?
- **(universal proper termination)** for a given requirement on the produced output data (resp. on the relation between input data and output data), does the workflow terminates for every possible input to the system ?
- **(existential proper termination)** for a given requirement on the produced output data (resp. on the relation between input data and output data), is there a particular input and at least one execution for which the workflow terminates ?

Universal termination provides a guarantee that a complex workflow will terminate and return a result for any input. This can be seen as a termination guarantee. Existential termination can be considered as a sanity check: a complex workflow that has no valid execution never terminates, regardless of input data, and should hence be considered as ill-formed. Universal proper termination guarantees that a workflow terminates and that it computes data that conforms to the client's requirement. Existential proper termination is also a sanity check, showing that a workflow is able to produce correct data for at least one of its executions.

3 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 [7], 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[10], but several types of applications such as opinion polls, citizen participation, etc. can be seen as complex crowdsourcing tasks.

3.1 Workflow ingredients

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 *atomic* or a *complex* task.

A simple competence model. We assume a fixed and finite pool $\mathcal{U}$ of workers, and an a priori finite list of competences **comp**. Each worker $u \in \mathcal{U}$ can complete or refine some tasks according to its *skills*. We hence define a map $sk : \mathcal{U} \to \mathbf{comp}$. Notice that $sk(u)$ is a set, i.e. a particular competence $c \in sk(u)$ needs not be exclusive. We adopt this simplistic model of workers competences for clarity of the model, but more evolved representations of skills exists and could be easily integrated to the model. For instance, [24] proposes a hierarchy of competences to reflect a natural ranking of the expertise. However, in this paper, we do not consider skills classification nor management of competences, and just view skills of a worker as a set of keywords.

[10] http://zoo1.galaxyzoo.org/

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 an 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 tasks* 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 $T_{cs} : \mathcal{T} \to 2^{\mathbf{comp}}$, depicting the fact that a worker u is allowed to realize or refine task t if it has the required competences, i.e., if $T_{cs}(t) \cap sk(u) \neq \emptyset$. This competence model is not essential for the (un)decidability results presented hereafter in the model. One could indeed consider that every worker has all competences, and can perform any task within the system. Within this setting, one needs not define workers competences, nor attach skills constraints to tasks. However, for practical use of a crowdsourcing platform, one usually wants to obtain the best possible results, which calls for a clever management of skills, incentives, etc.

Let us now explain how task refinement is modeled. Let $t \in \mathcal{T}_{cx}$ be a complex task, and let $C_t = T_{cs}(t)$ be the set of competences allowing a worker to complete successfully t. As already explained, we advocate that crowdsourcing platforms should allow higher-order answers. So, we assume that, as soon as a worker u owns one appropriate skill to solve task t, he is able to refine it: either he already knows a very generic way to decompose tasks of this kind, or the platform allows any higher-order answer depicted as a finite workflow to complete the task. In the first case, the workflow to complete t is fixed a priori (it can be for instance a generic map-reduce pattern to distribute data and aggregate results). In the second case, the higher-order answer belongs to a finite set of possible workflows (for instance workflows of bounded size assembling nodes which labels belong to a finite alphabet of tasks). Such possibility is already in use in the world of crypto trading (platforms such as Kryll allow users to define simple trading bots using a block diagram language [**?**]). In the following, we hence assume that a competent worker possesses several finite orchestrations depicting appropriate refinements of t. We will denote by $Profile(t, u)$ this finite set of finite workflows. 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. Note that a profile in $Profile(t, u)$ needs not be a workflow of large size, and may even contain workflows with a single node. In this case, the refinement simply replaces $t \in \mathcal{T}_{cx}$ by a single atomic tagging task $t' \in \mathcal{T}_{ac}$, meaning that u thinks that the task is easy, and wants it to be realized by a single worker.

Let us consider the following example: a complex task t_{like} asks to rank large collections of images of different animals with a score between 0 and 10. The relational schema for the dataset D used as input for t_{like} is a collection of records of the form $Picdata(nb, name, kind)$ where nb is a key, $name$ is an identifier for a picture, kind the species represented on the picture obtained from former annotation of data by crowdworkers. A worker u can decide to divide dataset D into three disjoint datasets containing pictures of cats, pictures of dogs, and pictures of other animals. The contents of these datasets can be ranked separately, and then the results of ranking aggregated. Figure 2 represents a possible profile to refine task t_{like}. A task t_{like} is rewritten in a workflow with four nodes. Node n_0 is an occurrence of an automated tasks that splits the original dataset into datasets containing pictures of dogs, cats, and other animals. Nodes n_1, n_2, n_3 are occurrences of tagging tasks for the respective animal kinds, and node n_f is an occurrence of an automated task that aggregates the results obtained after realization of preceding tasks.

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.) [8]. In this paper, we leave this notion of incentives apart, and consider that all workers

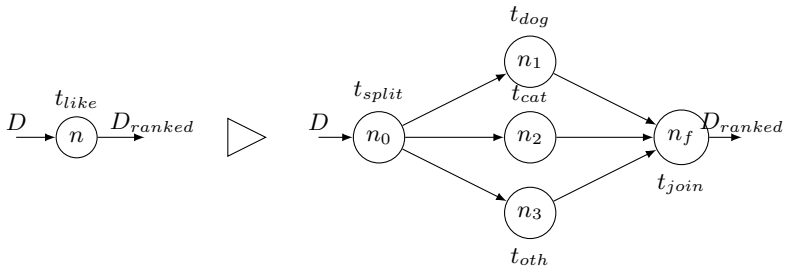

Fig. 2. A profile for refinement of task t_{like}.

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.

3.2 Workflows

Definition 1 (Workflow). *A* workflow *is a labeled acyclic graph $W = (N, \longrightarrow, \lambda)$ where N is a finite set of nodes, representing occurrences of tasks, $\longrightarrow \subseteq N \times N$ is a precedence relation, and $\lambda : N \to \mathcal{T}$ 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 $\mathcal{U}$ and $\mathcal{T}$ are fixed, and we will denote by $\mathcal{W}$ the set of all possible workflows. Intuitively, if $(n_1, n_2) \in \longrightarrow$, then an occurrence of task named $\lambda(n_1)$ represented by n_1 must be completed before an occurrence of task named $\lambda(n_2)$ represented by n_2, and that data computed by n_1 is used as input for n_2. We denote $min(W)$ the set of sources of W, by $succ(n)$ the set of successors of a node n, and by $pred(n)$ 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. The property prevents from launching tasks which results are never used to build an answer to a client.
- for every workflow $W = (N, \longrightarrow, \lambda) \in Profile(t, u)$, the labeling λ is injective. This results in no loss of generality, as one can create copies of a task for each node in W, but simplifies proofs and notations afterwards. Further, W has a unique source node $src(W)$.

Definition 2 (Refinement). *Let $W = (N, \longrightarrow, \lambda)$ be a workflow, $W' = (N', \longrightarrow', \lambda')$ be a workflow with a unique source node $n'_{src} = src(W')$ 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 $W_{[n/W']} = (N_{[n/W']}, \longrightarrow_{[n/W']}, \lambda_{[n/W']})$, where:*

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

To illustrate the notion of refinement, consider the example of Figure 3. In the workflow at the left of the Figure, node n_1 is replaced by the profile of Figure 2 for taks t_{like}. The result is the the workflow on the right of Figure 3.

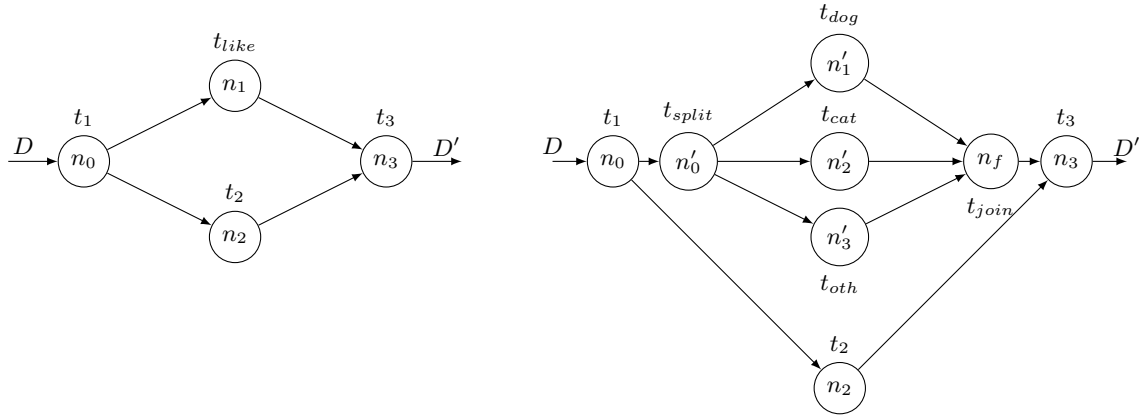

Fig. 3. A refinement of node n_1, replaced by the profile for task t_{like} in Figure 2.

3.3 Data

The data 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. We use a standard relational model to represent data [6], i.e., data is organized in datasets, that follow *relational schemas*. We assume finite set of domains $\mathbf{dom} = dom_1, \ldots, dom_s$, a finite set of attribute names **att** and a finite set of relation names **relnames**. Each attribute $a \in \mathbf{att}$ is associated with a domain $dom(a) \in \mathbf{dom}$. 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. A *record* of a relational schema $rs = (rn, A)$ is tuple $rn(v_1, \ldots v_{|A|})$ where $v_i \in dom(a_i)$ (it is a row of the table), and a *dataset* with relational schema rs is a multiset of records of rs. A *database schema* DB is a non-empty finite set of tables, and an instance over a database DB maps each table in DB to a dataset.

Complex workflows manipulate data, either automatically via automated tasks, or with the help of workers. Seen as data transformations, tasks can be seen as mechanisms that apply FO queries to datasets, add new tuples from a given domain to an existing dataset, add a new field to every record in a dataset, etc. 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 $\mathcal{D}^{in} = D_1^{in}, \ldots D_k^{in}$. 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 $\mathcal{D}^{out} = D_1^{out} \ldots D_q^{out}$. As for inputs, we will consider that dataset D_i^{out} is the data sent to the i^{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.

Consider for instance the example of Figure 2, and in particular node n_0 that represents an automated task which automatically splits a dataset into three smaller datasets D_{cats}, D_{dogs}, D_{oth}. Datasets $D, D_{dogs}, D_{cats}, D_{oth}$ use the relational schema $rs = Picdata(nb, name, kind)$. Further, dataset D_{dogs} can be obtained as a simple restriction of D to tuples such that $kind =$ "dog". This is easily encoded as an SQL formula. Conversely, D and D_{dogs} satisfy the FO formula $\forall nb, n, k, (Picdata(nb, n, k) \in D \wedge k =$ "dog") $\Leftrightarrow$

$Picdata(nb, n, k) \in D_{dogs}$. 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 operations : projections on a subset of attributes, selection of records that satisfy some predicate, record insertion or deletion.

To simplify workflow 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 $\mathcal{D}^{in}$ nor to define distinct output datasets. To refer to the way each D_i^{out} is computed, we will denote by $spl_t^{(i)}$, the function that associates to $\mathcal{D}^{in}$ the i^{th} output produced by a split task t (i.e. $spl_t^{(i)}(\mathcal{D}^{in}) = D_i^{out}$). 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, whose 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 $f_t(\mathcal{D}^{in}) = \bigcup_{i \in 1..|\mathcal{D}^{in}|} D_i$. 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 worker u, data D_{in} comes from preceding nodes, but the output depends on the worker. Hence, execution of task t by worker u produces an output $\mathcal{D}^{out}$ chosen non-deterministically from a set of possible outputs $F_{t,u}(\mathcal{D}_{in})$. We will however allow for the modeling of some a priori knowledge and of constraints on the values chosen by workers. For the rest of the paper, we will assume that the legal contents of $F_{t,u}(\mathcal{D}_{in})$ is defined as a first order formula (possibly with arithmetic constraints) $\phi_{t,in,out}$ 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 $\mathcal{D}^{out} \in F_{t,u}(\mathcal{D}_{in})$.

Consider again the example of Figure 2, and in particular node n_1 that represents an occurrence of an atomic task which goal is to rank pictures of dogs. This task starts from a dataset D_{dogs} that uses relational schema $PicData(nb, name, kind)$ and outputs a dataset $D_{dogs,ranked}$ with relational schema $PicData(nb, name, kind, rank)$ where $rank$ is an integer in $[0, 10]$. We can relate tuples in D_{dogs} and $D_{dogs,ranked}$ with the following formula:

$$\forall nb, n, k, PicData(nb, n, k) \in D_{dogs} \Leftrightarrow \exists r, r \in [0, 10] \wedge PicData(nb, n, k, r) \in D_{dogs,ranked}$$

As r can take only a finite number of integer values, we can still express this formula with with relational statements and boolean connectives. In the case of split and merge operations, one can still define FO formulas relating datasets before and after the execution of a particular node. Consider the example of Figure 3(left), and in particular node n_3. Let us assume that this node performs the disjoin union of tuples in datasets D_1, D_2 produced by execution of tasks attached to nodes n_1 and n_2, and that both D_1, D_2 contain tuples of the form $PicData(nb, n, k, r)$. Let D_3 be the dataset produced by node n_3. Then the FO formula expressing the fact that D_3 is the disjoint union of D_1 and D_2 is:

$$\forall nb, n, k, r, PicData(nb, n, k, r) \in D_1 \wedge PicData(nb, n, k, r) \in D_2 \Leftrightarrow PicData(nb, n, k, r) \in D_3$$

Still on the example of Figure 2(left), assume that t_2 produces a dataset D_2 that is a selection of pictures by a separate pool of workers, represented by tuples of the form $Selection(nb, score)$, where nb is the unique identifier of a picture, and that the role of task t_3 is to keep pictures that were both choosen in D_1 and D_2, and keep the maximal score attached to each picture in any of the datasets. Then, the relation among input D_1, D_2 and D_3 is :

$$\forall nb, n, k, r, nb', s, \begin{array}{l} PicData(nb, n, k, r) \in D_1 \wedge Selection(nb', s) \in D_2 \wedge nb = nb' \\ \Leftrightarrow PicData(nb, n, k, r') \in D_3 \wedge [(r > s \wedge r' = r) \vee (r \leq s \wedge r' = s) \end{array}$$

The dependencies between input data and output data of a particular task can hence be captured by FO formulas (with some arithmetic constraints): either tasks apply SQL-like queries and the relation between

input and outputs can be captured by First-Order logic [6]. We will show in section 6 that these dependencies can be used to prove feasibility of a particular execution of a complex workflow, even when manipulated data are not precisely known.

4 Operational semantics

In section 4, we have introduced all ingredients to define formally complex workflows and their semantics:

Definition 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,* $\mathcal{P} : \mathcal{T} \times \mathcal{U} \to 2^{\mathcal{W}}$ *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*.*

Intuitively, in a complex workflow, W_0 is an initial high-level description provided by a requester. The rest of the description is the execution environment for this complex process: it defines a *finite* set of users, their skills, and their knowledge in terms of task decomposition.

The execution of a complex workflow consists in realizing all its tasks, following the order given by the dependency relation $\longrightarrow$ 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 $W = (N, \longrightarrow, \lambda)$ is a partial map $Ass : N \to \mathcal{U}$ 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 $\mathbf{Dass} : N \to (DB \uplus \{\emptyset\})^*$, that assigns a sequence of input datasets to nodes in W. For a node with k predecessors $n_1, \ldots n_k$, we have $\mathbf{Dass}(n) = D_1 \ldots D_k$. 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 4 (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 final output data. Note that due to data assignment, there can be more than one final configuration, and we denote by $\mathcal{C}_f$ the set of all final configurations.

We define the operational semantics of a complex workflow with 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.

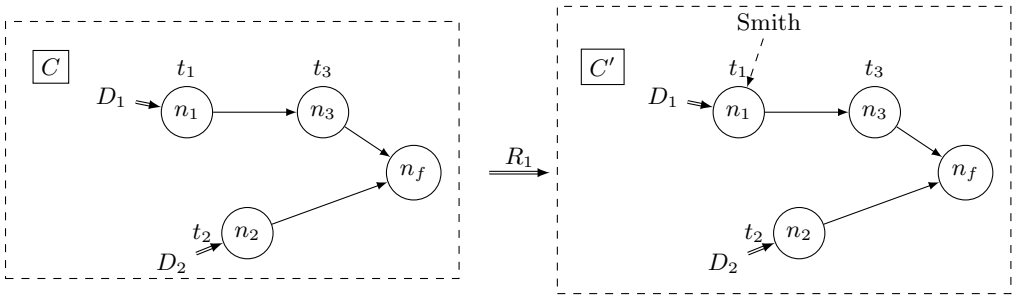

Fig. 4. Application of semantic rule R_1

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

Consider for instance the application of rule R_1 described in Figure 4. Configurations are represented by the contents of dashed rectangles. Workflow nodes are represented by circles, tagged with a task name representing map λ. The dependencies are represented by plain arrows between nodes. Worker assignments are represented by dashed arrows from a worker name u_i to its assigned task. Data assignment are represented by double arrows from a dataset to a node. The left part of Figure 4 represents a configuration C with four nodes n_1, n_2, n_3 and n_f. The predecessors of n_1 and n_2 have been executed. Node n_1 represents occurrence of a task of type t_1 and is attached dataset D_1. Let us assume that D_1 is a database containing *bee* pictures, and that task t_1 cannot be automated ($t_1 \notin$) and consists in tagging these pictures with bee names, which requires competences on *bee* species. Let us assume that worker *Smith* is currently not assigned any task and has competences on bees. Then $sk(Smith) \cap T_{cs}(t_1) \neq \emptyset$ and rule R_1 applies. The resulting configuration is configuration C' at the right of the Figure, where the occurrence of t_1 represented by node n_1 is assigned to worker Smith.

Rule 2 (Atomic Task Completion): An atomic task $t = \lambda(n)$ 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 available.

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

Consider the example of Figure 5. We start from a configuration in which worker *Smith* has to tag images stored in a dataset D_2. We assume that the relational schema for D_2 is a tuple $R(id, pic)$ where id is a key and pic a picture. We also assume that tags are species names from a finite set of taxons, e.g. $Tax = \{Honeybee, Bumblebee, Masonbee, ..., Unknown\}$. Worker Smith performs the tagging task, which results in a dataset D_3 with relational schema $R'(id, pic, tag)$. One can notice that no information is given of the way worker Smith tags the pictures in D_2, the only insurance is that for every tuple $R(id, pic)$, there exists a tuple $R'(id, pic, t)$ in D_3 where $t \in Tax$. Notice that application of this rule may results in several successor configurations, as each tag attached to a tuple is a non-deterministic choice of the worker realizing the task.

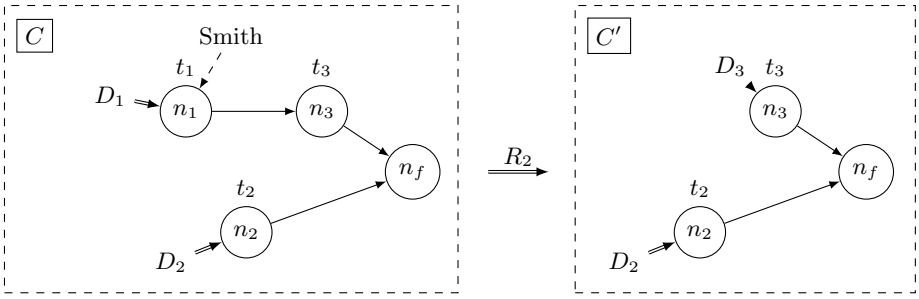

Fig. 5. Application of semantic rule R_2

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.

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

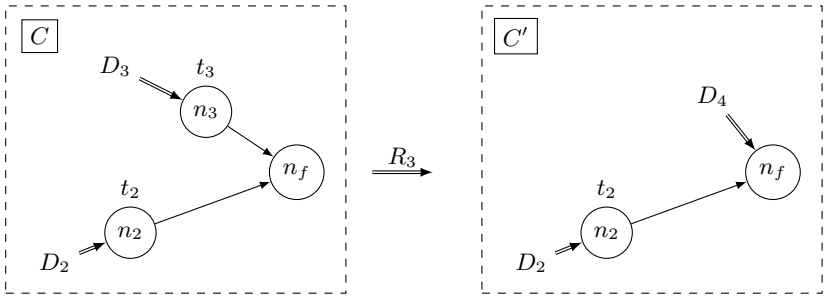

Fig. 6. Application of semantic rule R_3

Consider the example of Figure 6. We resume from the situation in Figure 5, i.e. with two nodes n_2, n_3 remaining to be executed before n_f, and with a dataset composed of tagged images attached to node n_3. Let us assume that task t_3 is an automated task that consists in pruning out images with tag "unknown". This task can be realized as a projection of D_3 of tuples $R'(id, pic, t)$ such that $t \neq "Unknown"$. As a result, we obtain a dataset D_4, used as input by node n_f such that $\forall R'(id, pic, t) \in D_3 \wedge t \neq "unknown", \exists R'(id, pic, t) \in D_4$ and the task can be realized by simple SQL query.

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 $W_s = (N_s, \longrightarrow_s, \lambda_s) \in Profile(t, u)$. 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.

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

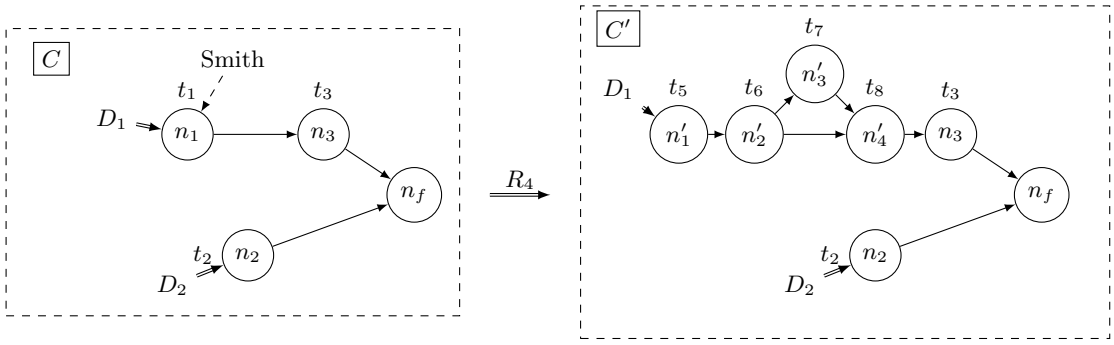

Fig. 7. Application of semantic rule R_4

Consider the example of Figure 7. Let us assume that Worker "Smith" is assigned task t_1 and that this task is a complex tagging task (for instance workers are asked to find names of rare species). In such situation, Smith can decide to replace the task by a simple single-worker tagging mechanism, or by a more complex workflow, that asks a competent worker to tag pictures, then separates the obtained datasets into pictures with/without tag "Unknown", and sends the *unknown* species to an expert (for instance an entomologist) before aggregating the union of all responses. This refinement lead to a configuration C', shown in the right part of Figure 7, where n'_1 is a tagging task, n'_2 is an automated task to split a dataset, n'_3 is a tagging tasks which requires highly competent workers and n'_4 is an aggregation task. In our model, we assume that refinement is always performed by a competent worker, owning an appropriate profile to handle the refinement.

Note that the definition of a *complex* task is very subjective and varies from one worker to another. Classifying tasks as complexor not a priori should not be seen as a limitation, as 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 $t' \in \mathcal{T}_{ac} \cup \mathcal{T}_{aut}$ if this possibility appears in her profile. This allows to model situations where a worker refines a task because she thinks it is too complex to be handled by a single person.

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 5 (Run). *A run $\rho = C_0.C_1 \dots C_k$ of complex workflow CW with workers $\mathcal{U}$ 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.*

Figure 8 gives an example of run. The top-left part of the figure is an initial 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.

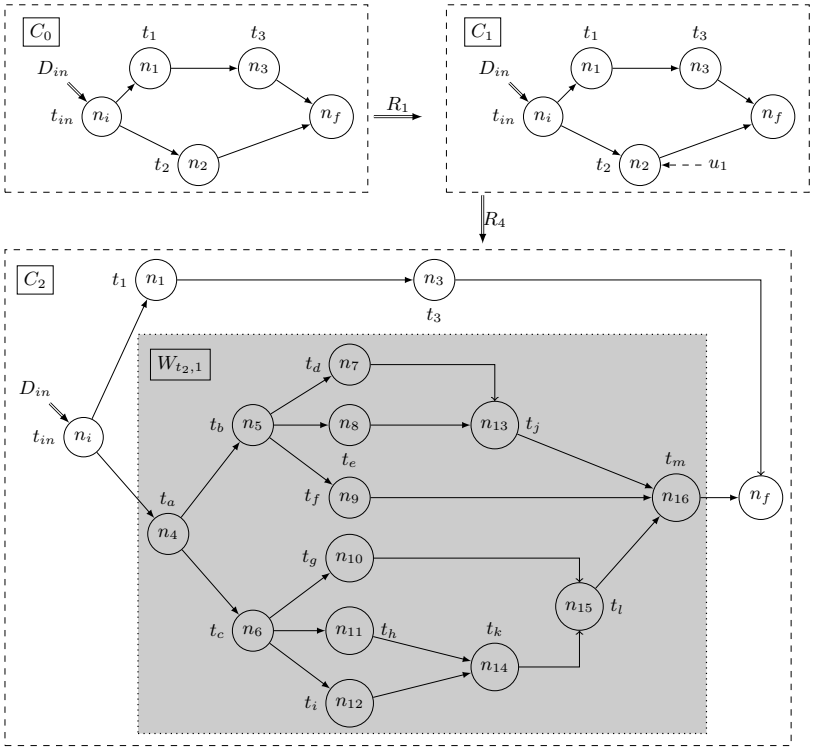

Fig. 8. Complex workflow execution. C_0 represents the initial configuration with data D_{in} allocated to node n_i. C_1 is the successor of C_0: user u_1 is allocated to node n_2, and $t_2 = \lambda(n_2)$ is a complex task. C_3 depicts the configuration after refinement of node n_2 by a new workflow W_{t2} (shown in the Grey rectangle).

In the rest of the paper, we denote by $\mathcal{R}uns(CW, D_{in})$ the set of maximal runs originating from *initial configuration* $C_0 = (W_0, Ass_0, \mathbf{Dass}_0)$ (where $\mathbf{Dass}_0$ associates dataset D_{in} to node n_i). We denote by $\mathcal{R}each(CW, D_{in})$ 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, $\mathcal{R}each(CW, D_{in})$ and $\mathcal{R}uns(CW, D_{in})$ need not be finite. Even when $\mathcal{R}each(CW, D_{in})$ 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 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 $\lambda(n_1) = t_1$ such that $\longrightarrow = \{(n_i, n_1), (n_1, n_f)\}$. 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 $\longrightarrow = \{(w_2, w_1), (w_1, w_f)\}$. Then, after application of rule $R1$ (assigning user 1 to the node that carries task t_1) and $R4$ (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 workfow 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.

5 Data and First order

We have introduced the standard data representation via relational schemas in section 3.3, i.e. a representation of records by tuples of the form $rn(a_1, \ldots, a_n)$, where rn is a relation name and $a_1, \ldots a_n$ attributes that fulfill the constraints of the legal domain of relation rn. As detailed in section 4, a client of a complex workflow can give constraints to reduce the range of legal input data, or constraints on expected outputs of a complex workflow. Similarly, we show in section 6 that deciding termination needs to consider properties of datasets contents during configuration changes. In the rest of the paper, we will use *First Order Logic* (FO) to address properties of datasets.

5.1 Decidability for FO fragments

Definition 6 (First Order). *A* First Order formula *(in prenex normal form) over a set of variables X is a formula of the form $\phi ::= \alpha(X).\psi(X)$ where $\alpha(X)$ is an alternation of quantifiers and variable names in X, i.e. sentences of the form $\forall x_1 \exists x_2, \ldots$ called the* prefix *of ϕ and $\psi(X)$ is a quantifier free formula called the* matrix *of ϕ. $\psi(X)$ is a boolean combinations of* atoms *of the form $R_i(x_1, \ldots x_k)$, $P_j(x_1, \ldots x_n)$, where $R_i(x_1, \ldots x_k)$'s are relational statements, and $P_j(x_1, \ldots x_n)$'s are predicates, i.e. boolean function on $x_1, \ldots x_n$.*

In the rest of the paper, we consider variables that either have finite domains or real valued domains, and predicates specified by simple linear inequalities. In particular we will consider equality of variables, i.e. statements of the form $x_i = x_j$. This class of constraints in well known, and deciding whether there exists an assignment satisfying such constraint can be done in polynomial time [14]. When a constraint over n variables is an expression of the form $\bigwedge x_i - x_j \leq c \wedge \bigwedge x_i \leq c$ where x_i, x_j are variables, and c a constant, it can be encoded as differential bound matrix (DBM), and emptiness of the domain represented by such a DBM where variables a real valued can be checked in $O(n^3)$.

Letting $X_1 = \{x_1, \ldots x_k\} \subseteq X$ we will often write $\forall \vec{X}_1$ instead of $\forall x_1 . \forall x_2 \ldots \forall x_k$. Similarly, we will write $\exists \vec{X}_1$ instead of $\exists x_1 . \exists x_2 \ldots \exists x_k$. Given an FO formula in prenex normal form, we will use w.l.o.g. formulas of the form $\forall \vec{X}_1 \exists \vec{X}_2 \ldots \psi(X)$ or $\exists \vec{X}_1 \forall \vec{X}_2 \ldots \psi(X)$, 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*. By allowing blocks of arbitrary size, and in particular empty blocks, this notation captures all FO formulas in prenex normal form. We denote by $\phi_{[t_1/t_2]}$ the formula obtained by replacing every instance of term t_1 in ϕ by term t_2. Each variable x_i in X has its own domain $Dom(x)$. A variable assignment (for a fixed set of variables X) is a function μ that associates a value d_x from $Dom(x)$ to each variable $x \in X$. Given a variable assignment μ, we say that ϕ hold under μ, and write $\mu \models \phi$ iff ϕ evaluates to *true* under assignment μ.

Definition 7 (satisfiability). *A variable free formula is* satisfiable *iff it evaluates to true. A formula of the from $\exists x, \phi(x)$ is* satisfiable *if and only if there is a way to choose a value for x such that $\phi(x)$ is satisfied, i.e. there exists a value $d_x \in Dom(x)$ such that $\phi(x)_{[x/d_x]}$ is satisfiable. A formula of the form $\forall x, \phi(x)$ is true if and only if, for every possible choice of a value $d_x \in Dom(x)$ for x, $\phi(x)$ is satisfiable, i.e $\phi(x)_{[x/d_x]}$ is satisfiable.*

It is well known that satisfiability of first order logic is undecidable in general, but it is decidable for several fragments. The *universal fragment* of FO is the set of formulas of the form $\forall \overrightarrow{X_1}\, \phi$, where ϕ is quantifier free. Similarly, the existential fragment of FO contains only formulas of the form $\exists \overrightarrow{Y_1}\, \phi$. Consider again our setting, where predicates in FO formula can be encoded by relations and boolean predicates that are linear inequalities. Then checking satisfiability of the existential/universal fragment of FO can be done non-deterministically in polynomial time.

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

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

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

One needs not restrict to existential or universal fragments of FO to decidability of satisfiability. A well known decidable fragment is (FO^2), that uses only two variables [26]. However, as this fragment forbids in particular atoms of arity greater than 2, which is a severe limitation when addressing properties of datasets. The BS fragment of FO is the set of formulas of the form $\exists \overrightarrow{Y}_1 . \forall \overrightarrow{X}_2 . \psi$, where ψ is quantifier free, may contain predicates, but no equality. The *Bernays-Schonfinkel-Ramsey* (BSR) fragment of FO [5] extends the BS fragment by allowing equalities in the matrix ψ. Satisfiability of a formula in the BS or BSR fragment of FO is NEXPTIME-complete [20]. Algorithms to check satisfiability for a fragments of FO can be obtained by transforming formulas from that class into an equivalent or equisatisfiable formulas of a decidable subclass. Two FO formulas ϕ and ψ are *equivalent* iff, for every variable assignment μ, $\mu \models \phi$ iff $\mu \models \psi$. They are equisatisfiable iff an assignment μ such that $\mu \models \phi$ exists iff an assignment μ such that $\mu' \models \psi$ exists. Transformation of this type are usually expensive, and result in a blowup of the size of considered formulas.

Recent results [30] exhibited a new fragment, called the *separated fragment* of FO, defined as follows: Let $Vars(A)$ be the set of variables appearing in an atom A. We say that two sets of variables $Y, Z \subseteq X$ are separated in a quantifier free formula $\phi(X)$ iff for every atom A of $\phi(X)$, $Vars(A) \cap Y = \emptyset$ or $Vars(A) \cap Z = \emptyset$. A formula in the *Separated Fragment* (SF) of FO is a formula of the form $\forall \overrightarrow{X}_1 . \exists \overrightarrow{Y}_2 . \dots \forall \overrightarrow{X}_n . \exists \overrightarrow{Y}_n \phi$, where $\overrightarrow{X}_1 \cdots \cup \overrightarrow{X}_n$ and $\overrightarrow{Y}_1 \cdots \cup \overrightarrow{Y}_n$ are separated. The SF fragment is reasonably powerful and subsumes the *Monadic Fragment* [23] (where predicates can only be unary) and the BSR fragment. Every separated formula can be rewritten into an equivalent BSR formula (which yields decidability of satisfiability for SF formulas), but at the cost of an n-fold exponential blowup in the size of the original formula. Satisfiability of a separated formula ϕ is hence decidable [30], but with a complexity in $O(2^{\downarrow n^{|\phi|}})$.

A recent and 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 [13]. In the rest of the paper, we will show classes of complex workflows for which termination and proper termination are decidable. All the results demonstrated in our paper would hold in a FO^2BD setting. However, complex worklfows handle imprecise worker inputs, and imprecise values of such fields are better captured with unbounded domains.

We will show hereafter that the universal fragment of FO suffices to encode conditions needed for basic operation to produce an empty dataset. Similarly, we will show that the universal fragment sufficed to encode the weakest preconditions required to satisfy an universal formula. This will be of particular importance to prove decidability of termination.

5.2 Closure of FO classes

In section 6, we will give an algorithm to check termination of a complex workflow with bounded recursion. Roughly speaking, this algorithms searches a reachable configuration C_{bad} where emptiness of a dataset D could stop an execution. Once such a configuration is met, it remains to show that the statement $D = \emptyset$ is compatible with the insertion, projections, unions of datasets performed during the execution before reaching C_{bad}. This is done by computing backward the weakest preconditions ensuring $D = \emptyset$ along the followed run, and checking that each condition is satisfiable.

Definition 8. *Let $C \longrightarrow C'$ be a move from configuration C to C' of a complex workflow. Let m be the nature of this move (an automated task realization, an user assignment, a refinement,...). We denote by $wp[m]\psi$ the* weakest precondition *required for C such that ψ holds in C after move m.*

Weakest precondition were introduced in [10] as a way to prove correctness of programs. In a program written in an imperative language, if the property ψ is an inequality of the form $\psi_1 ::= x \leq 10$ and m the instruction $x := x + 1$, then $wp[x := x + 1]\psi_1$ is the property $x \leq 9$. Calculus of weakest precondition was also proposed to verify web applications with embedded SQL [13].

Proposition 2. *Let CW be a complex workflow, r be the maximal arity of relational schemas in CW and ψ be an FO formula. Then for any move m of CW, $wp[m]\psi$ is effectively computable, and is of size in $O(r.|\psi|)$.*

Proof (sketch). The effect of moves on the contents of datasets can be described as sequential composition of basic operations that are projections of datasets, insertions of records or fields, unions or joins of dataset. Let $D_1, \dots D_k$ be the datasets that have to satisfy ψ. If some D_i is obtained as a projections of a dataset D'_i, then D_i contains only records of D'_i satisfying some predicate P. The precondition will hence be obtained by a simple replacement in ψ of any statement of the form $rs(\overrightarrow{x}) \in D_i$ by $rs(\overrightarrow{x}) \in D_i \wedge P$.

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

Similarly, if some dataset D_i is obtained as the union of two datasets D'_1, D'_2, the precondition for ψ should consider, for every statement of the form $rs(\overrightarrow{x}) \in D_i$, cases where $rs(\overrightarrow{x})$ belongs to D_1 and cases where $rs(\overrightarrow{x})$ belongs to D_2. These two cases are exclusive, and as tuples identified by $\overrightarrow{x}$ have to appear in at least one dataset (either D_1 or D_2) we simply replace atoms of the form $rs(\overrightarrow{x}) \in D_i$ by the disjunction $rs(\overrightarrow{x}) \in D_1 \vee rs(\overrightarrow{x}) \in D_2$. We also replace negative statements $rs(\overrightarrow{x}) \notin D_i$ by conjunction $rs(\overrightarrow{x}) \notin D_1 \wedge rs(\overrightarrow{x}) \notin D_2$. The size of the obtained formula is hence in $O(2.|\psi|)$.

If some D_i is obtained by creation of a field for each record in dataset D'_i, then relational statement $rs(\overrightarrow{x}) \in D_i$ is replaced by another statement $rs(\overrightarrow{y}) \in D'_i \wedge P'(\overrightarrow{y})$ where $\overrightarrow{y}$ is a subset of $\overrightarrow{x}$, and $P'(\overrightarrow{y})$ is a quantifier free predicate indicating constraint on $\overrightarrow{y}$ obtained after variable elimination when $\overrightarrow{x}$ takes legal values imposed by relational schema of D_i (i.e. it satisfies $Ldom_i$–see proof of Proposition 1–) and satisfies the constraints of relational schema of D'_i.

For joins, relation of the form $rs(\overrightarrow{X}_i)$ are transformed in statements of the form $rs(\overrightarrow{Y_i}) \wedge rs(\overrightarrow{Z_i}) \wedge y_1 = z_1$, where $y_1 \in \overrightarrow{Y_i}$ and $z_1 \in \overrightarrow{Z_i}$. This may multiply the size of the formula by r.

We refer interested readers to appendix A.5 for details on the construction of weakest preconditions. □

Computing a weakest precondition is mainly a syntactic replacement of a set of relational statements, or a disjunction of such replacements, that changes the number of variables. However, it does not change the number of quantifier blocks nor their ordering, and it does not introduce new quantifiers when replacing an atom. We hence easily obtain the following corollary:

Corollary 1. *The existential, universal, BSR and SF fragments of FO are closed under calculus of a weakest precondition in Complex Workflows.*

6 Termination

Complex workflow use the knowledge and skills of crowd workers to complete a task starting from input data provided by a client. However, a workflow may never reach a final configuration. This can be due to particular data input by workers that cannot be processed properly by the workflow, 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 $\mathcal{C}_f$.

In this section we address the questions of existential and universal termination. We first show that existential termination is undecidable, regardless of the inputs specified for a workflow. We then show that decidability and complexity of universal termination depends on the power allowed to specify inputs of the workflow.

6.1 Existential termination

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

When addressing the termination question for a set of inputs $\mathcal{D}_{in}$, we describe $\mathcal{D}_{in}$ symbolically with a decidable fragment of FO (e.g. the existential, universal, BSR or separated fragment of FO introduced in section 5). Termination questions are meaningful for clients and workflow designers. Existential termination ensures that an answer to a client (an output dataset D_{out}) *can be returned*. This is an essential sanity check for a particular input D_{in}: if a complex workflow never returns an answer when using D_{in} as input, then this dataset can be the cause of the problem. If a complex workflow CW does not terminate for a set of inputs $\mathcal{D}_{in}$ depicting possible entries provided by a client, then CW can be considered as ill-formed. Similarly, universal termination provides guarantees on the correctness of CW. If CW terminates for all inputs, then every input of a client will be processed and the computed answer returned. This does not yet means that the computed data is always correct, but at least that the execution of a workflow will terminate for this input or all legal inputs provided by the client. We will show in this section that existential termination is undecidable, and universal termination is in $co - NEXPTIME$.

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

Proof (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 worker 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, worker u_i has to guess whether the value of a counter is zero or not (this is encoded as a choice of a particular workflow to refine t_i). If the worker does the wrong guess, the execution deadlocks. Otherwise, the execution always proceeds to the next instruction. More details on the encoding are provided in Appendix A.1.

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

Remark 1. Notice that the undecidability of existential termination does not arise from complex manipulation of data by workers nor by automated tasks. This undecidability occurs as soon as higher order operations are allowed, and distribution of datasets to successor nodes is allowed for non-empty datasets only.

6.2 Universal Termination

Universal termination is somehow an easier problem than existential termination. We show in this section that it is indeed decidable for many cases and in particular when the datasets used as inputs of a complex workflow are explicitly given or are specified in a decidable fragment of FO. We proceed in several steps. We first define symbolic configurations, i.e. descriptions of the workflow part of configurations decorated with relational schemas depicting data available as input of tasks. We define a successor relation among these symbolic configurations. We then define the class of non-recursive complex workflows, in which all executions are finite, and give a bound $K_{\mathcal{T}_{cx}}$ on the length of executions in non-recursive specifications. The next step is to show that given a symbolic executions ρ^S of length n, and a description of inputs, the complexity of checking whether there exists an execution ρ that coincides with ρ_S is at least in $2EXPTIME$. This proof builds on the iterative calculus of weakest preconditions needed to allow a particular run. The complexity originates mainly from the growth of formulas depicting weakest preconditions at each iteration. Deciding universal termination builds on the same construction. We show that recursive specifications do not terminate universally (they either deadlock or continue forever). This gives a bound on the length of runs that have to be considered for recursion-free complex workflows, and allows to prove that the complexity of universal termination is at least in $co-2EXPTIME$.

Let us first define symbolic configurations. Roughly speaking, a symbolic configuration describes the status of workflow execution as in standard configurations (see defn. 4) but leaves the data part underspecified.

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

For every node n that is minimal in W, the meaning of $\mathbf{Dass}^S(n) = rs_1, \dots rs_k$ is that task attached to node n takes as inputs datasets $D_1 \dots D_k$ where each D_i conforms to relational schema rs_i. Notice that it is sufficient to know $\lambda(n)$ to obtain $\mathbf{Dass}^S(n)$. A symbolic configuration $C_j^S = (W_j, Ass_j, Dass_j^S)$ is the *successor* of a symbolic configuration $C_i^S = (W_i, Ass_i, Dass_i^S)$ iff one of the following situation 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 $n \in min(W_i)$ such that $t = \lambda(n)$ is an automated task (resp. an atomic task) manipulating datasets $D_1, \dots D_q$, 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 $f_t(D_1, \dots D_q)$ (resp. $F_t(D_1, \dots D_q)$).
- there exists $n \in W_i$, $\lambda(n)$ is a complex task, and $Ass(n) = u$, W_j is the workflow obtained by replacement of n in W_i by a workflow $W^{new} \in \mathcal{P}(\lambda(n), u)$. $\mathbf{Dass}^S$ assigns to the copy of minimal node n_j of W^{new} the relational schemas in $Dass_i^S(n)$.

We denote by $SC(C_i^S)$ the set of successor configurations of symbolic configuration C_i^S.

Definition 11 (Deadlocks, Potential deadlocks). *A symbolic configuration $C^S = (W, Ass, Dass^S)$ 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, i.e. there exists $n, n_1, n_2 \in W$ such that n is minimal and $\{(n, n_1); (n, n_2)\} \subseteq \longrightarrow_W$.*

A deadlocked symbolic configuration represents a situation where a workflow progress is blocked due to shortage of competent users to execute tasks. These configurations should be avoided to terminate the execution of a complex workflow. A potential deadlock is a symbolic configuration from which absence of data in a particular dataset may stop an execution. We use the term potential deadlock because one cannot

know from a symbolic configuration whether a particular dataset D_i is empty. We will show in this section that one can decide whether a potential deadlock situation in C^S represents a real and reachable deadlock, by considering how the contents of dataset D is forged along the execution leading to C^S.

Definition 12 (Symbolic execution). *A* symbolic execution *is a sequence* $\rho^S = C_0^S \xrightarrow{m_1} C_1^S \xrightarrow{m_2} \dots \xrightarrow{m_k} C_k^S$ *where each* C_i^S *is a symbolic configuration, and:*

- $C_0^S = (W_0, Ass_0, \mathbf{Dass}^S)$ *where* W_0, Ass_0 *have the same meaning as for configurations, and* $\mathbf{Dass}^S$ *associates to the minimal node* n_0 *in* W_0 *the relational schema of* $Ass_0(n_0)$.
- C_{i+1}^S *is a successor of* C_i^S.

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

Proposition 3. *Let* CW *be a complex workflow,* D_{in} *be a dataset,* $\mathcal{D}_{in}$ *be an FO formula with* n_{in} *variables, and* $\rho^S = C_0^S \dots C_i^S$ *be a symbolic workflow. Then deciding if there exists a run* ρ *with input dataset* D_{in} *and signature* ρ^S *is in* $2EXPTIME$. *Checking if there exists a run* ρ *of* CW *and an input dataset* D_{in} *that satisfies* $\mathcal{D}_{in}$ *with signature* ρ^S *is*

- *undecidable in general.*
- *in* $2EXPTIME$ *if* $\mathcal{D}_{in}$ *is in the universal/existential fragment of FO, or in the BSR fragment of FO,*
- $n_{in} - foldEXPTIME$ *in the size of* $\mathcal{D}_{in}$ *if* $\mathcal{D}_{in}$ *is in the separated fragment of FO*

Proof. We check feasibility of ρ^S, that is starting from C_i^S, check that all conditions met along a run with signature ρ^S to reach a configuration C_i that is compatible with C_i^S are met at each step, and allow for existence of configurations $C_0, C_1, \dots C_{i-1}$. First notice that the actual run with signature ρ^S performs the same sequence of moves as in ρ^S, and that the question of existence of a run ρ with signature ρ^S only questions satisfiability of constraints on data computed at each step of this run, not the sequence of moves along ρ. Second, one can notice that if ρ^S contains a deadlock, it is necessarily the last symbolic configuration of the run. So one needs not check existence of a deadlock separately when checking feasibility of ρ^S. A third remark is that semantic rules that affect users to tasks or perform a refinement do not consider data contents. Hence, if the move from C_{i-1} to C_i is an user assignment or a refinement, then it is necessarily feasible as long as C_{i-1} is reachable. The only cases where data can affect execution of a step along a run is when an automated task or an atomic task has to process empty data. For each of these steps, one has to check that the inputs of an executed task are not empty, i.e. suppose that $D_1 \neq \emptyset \wedge \dots \wedge D_k \neq \emptyset$. Non-emptiness of a dataset D_k at some configuration C_{i-1} is a property that depends on previous steps in the execution. For instance, if the move from C_{i-2} to C_{i-1} realizes the projection of a dataset, i.e. filters records in a dataset D'_k to keep only those that satisfy some predicate P, then the precondition that must hold at C_{i-2} is $\psi_{i-2} ::= \exists \overrightarrow{X}\ P(X)$. Similarly, if the move $C_{i-2} \xrightarrow{m_{i-1}} C_{i-1}$ is a transformation of records, a transformation of some dataset that adds new fields, a join of two datasets, the formula ψ_{i-2} is an existential FO formula.

This generalizes to the whole signature. Let ψ_k be an FO formula that has to be satisfied by configuration C_k in a run compatible with signature ρ^S. There exists a sequence of moves $C_0 \xrightarrow{m_1} C_1 \dots \xrightarrow{m_k} C_k$ iff the sequence $C_0 \xrightarrow{m_1} C_1 \dots \xrightarrow{m_{k-1}} C_{k-1}$ ends in a configuration C_{k-1} such that $C_{k-1} \models wp[m_k]\psi_k$ (by definition of weakest precondition). If $wp[m_k]\psi_k$ is not satisfiable, then the move from C_{k-1} to C_k *always* ends with datasets that do not fulfill ψ_k. One can decide whether $wp[m_k]\phi_k$ is satisfiable, as ψ_k is in a decidable fragment of FO, then by Prop. 1, one can effectively compute $wp[m_k]\psi_k$ and the obtained formula belongs to the same FO fragment as ψ_k. Now, in addition to the weakest preconditions for ψ_k to hold, one may have

to assume that several datasets are non-empty at stages $k-1$. Adding a statement $D_i \neq \emptyset$ at several steps of the signature just adds an existential conjuct, to all $\psi_i's$ are in the existential fragment of FO, which is preserved by weakest precondition computation (Prop. 1).

Now, one can build inductively all weakest preconditions $\psi_{k-1}, \psi_{k-2}, \ldots, \psi_0$ that have to be satisfied respectively by configurations $C_{k-1}, \ldots, C_0$. If any of these preconditions is unsatisfiable, then there exists no run with signature $C_0^S \ldots C_k^S$ leading to a configuration that satisfies ψ_k, and hence ρ^S is not the signature of an actual run of CW. One can notice also that, according to Prop. 1, the size of ψ_0 can be in $O(r^k)$ where r is the maximal arity of relational schemas of the complex workflow. Hence, checking all preconditions for a run of size k compatible with ρ^S can have a complexity that is in $O(2^{r^k})$.

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

Similarly, if $\mathcal{D}_{in}$ is given as a FO formula, the complexity depends on the considered fragment used to specify $\mathcal{D}_{in}$. In general, if $\mathcal{D}_{in}$ is given as a FO formula, it is undecidable if $\mathcal{D}_{in} \wedge \psi_0$ is satisfiable. If $\mathcal{D}_{in}$ is an existential/universal formula, then the complexity is exponential in the size of $\mathcal{D}_{in}$ and also exponential in the size of ψ_0. Assuming that $|\mathcal{D}_{in}| \leq 2^k$ we have a $2EXPTIME$ complexity. If $\mathcal{D}_{in}$ is in the BSR fragment, then checking satisfiability of $\mathcal{D}_{in}$ is in $2EXPTIME$, and so the overall process is in 2EXPTIME. Last if $\mathcal{D}_{in}$ is in the separated fragment, then checking its satisfiability is n_{in}-fold exponential in the size of $\mathcal{D}_{in}$, so the overall process of checking realizability of ρ^S has an n_{in}-fold exponential complexity. □

An execution $\rho = C_0 \ldots C_k$ 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 run that does not terminate is a run that either ends in a configuration that is not final and from which no rule can be applied, or an infinite run. A move from C_i to C_{i+1} leaves the number of nodes unchanged (application of worker 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(application of rule R_4). Only in this latter case, the number of nodes may increase. The set of possible transformations of W and Ass occurring from C is bounded. Further, semantic rule $R4$ is the only rule that creates new nodes in the workflow part of a configuration. So when the number of occurrences of rule $R4$ in a run is bounded, the number of applications of rules $R1$,$R2$,$R3$ and hence the size of (symbolic) executions is also bounded. Complex workflows that can exhibit infinite runs are hence specification with recursive rewriting schemes.

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

When a complex workflow is not recursion free, then some executions may exhibit infinite behaviors in which some task t_i is refined infinitely often. Clearly, this prevents termination. Such an infinite rewriting loop can be either stopped because it appears in a sequence of moves that deadlocks, or be executable, hence preventing reaching final configurations in which all tasks have been executed. We claim without proof the following property:

Proposition 4. *Complex workflows that are not recursion free have non-terminating executions.*

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

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

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

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

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

Proposition 7. *A complex workflow terminates universally iff the following conditions hold:*

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

Proof. If CW has recursive task rewriting, then there is a cycle in the rewriting graph $RG(CW)$ that is accessible from a task $t_0 = \lambda(n_0)$ appearing in W_0. Hence, there is an infinite run $\rho^\infty = C_0 \xrightarrow{a_1} C_1 \xrightarrow{r_1} C_2 \dots$ of CW which moves are only user assignments (a move) to a node of the current workflow at configuration C_i followed by a rewriting (r moves) that creates new instances of tasks, such that the sequence of rewritten task follows the same order as in the cycle of $RG(CW)$. Similarly, if CW terminates, then all runs are finite, and infinite runs of the form ρ^∞ cannot exist.

If CW can reach a deadlocked configuration, then by definition, it does not terminate. If all runs of CW terminate, then from any configuration, there is a way to reach a final configuration, and hence no deadlock is reachable. □

Condition i) can be easily verified, by checking existence As soon as the symbolic execution tree is finite, checking condition ii) is a simple exploration of symbolic configurations to find deadlocks. Checking condition iii) is more involved, as it requires assumptions on data. Let C_i^S be a reachable symbolic configuration, 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 $C_0 \dots C_i$ 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 a FO formula of the form $\psi_i ::= \nexists x_1, \dots x_{|A_j|}, rn_k(x_1, \dots x_{|A_j|}) \in D_k$. For ψ_i to hold in configuration C_i, inputs of minimal nodes in W_{i-1} may have to satisfy some constraint ψ_{i-1}. Depending on the nature of the move from C_{i-1} to C_i, the preconditions on inputs at step $i-1$ differ. Let m_i denote the nature of move from C_{i-1} to C_i. We denote by $wp[m_i]\psi_i$ the weakest precondition needed on inputs of minimal nodes in C_{i-1}^S such that ψ_i holds on C_i^S. We can show (Proposition **??**) 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 symbolic run with signature $\rho^S = C_0^S \dots C_i^S$ 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 ρ^S such that D_k is an empty dataset. If one end weakest precondition calculus on configuration C_0^S with a condition ψ_0 that is satisfiable, and satisfied by D_{in}, then such a run exists. We have proved in Proposition 4 that one can effectively compute the weakest precondition of an FO property, and furthermore that the universal, existential, BSR, and separated fragment of FO were closed under weakest precondition calculus.

Now, emptiness of a dataset D_k with relational schema $rs_k = (rn_k, A_k)$ can be encoded with the separated formula $\phi^{\emptyset}_{D_k} ::= \nexists x_1, \dots x_{|A_k|}, rn_k(x_1, \dots x_{|A_k|}) \in D_k$. This means that condition *iii*) in Proposition 7 can be effectively checked. Then proof of Proposition 3 immediately gives a non-deterministic algorithm to check existence of a deadlock during an execution of a workflow.

Theorem 2. *Let CW be a complex workflow, D_{in} be an input dataset, and $\mathcal{D}_{in}$ be an FO formula. Universal termination of CW on input D_{in} is in co$-2EXPTIME$. Universal termination of CW on inputs that satisfy $\mathcal{D}_{in}$ is:*

- *undecidable in general*
- *in $co-2EXPTIME$ if $\mathcal{D}_{in}$ is in the universal fragment of FO, or in the BSR fragment of FO,*
- *n_{in}-fold exponential in the size of $\mathcal{D}_{in}$ if $\mathcal{D}_{in}$ is in the separated fragment.*

Proof (sketch).

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

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

If this execution contains a potential deadlock at symbolic configuration C_i^S, then C_i is a configuration from which a particular dataset D must be split and must be empty to cause a deadlock. Before concluding that C_i^S can be a real deadlock, one has to check whether that there exists an actual real execution $C_0 \dots C_i$ such that property $D = \emptyset$ holds at C_i. We can show that the precondition ψ_{i-1} that needs to hold at C_{i-1} in order to obtain an empty dataset D at step C_i are either of the form $D' = \emptyset$ for some D' (this is the case if the move from C_{i-1} to C_i just adds a field to D' to obtain D) of a FO formula in the universal fragment. Then one can repeat the following steps at each step $k \in i-1, i-2, \dots$ up to C_0^S:

- compute $\psi_k = wp[m_k]\psi_{k+1}$. We know that this weakest precondition can be computed, and that ψ_k is still an universal formula of size in $O(r.|\psi_{k+1}|)$ (see proposition 1).
- Check satisfiability of ψ_k. If the answer is false, then Fail: one cannot satisfy the conditions required to have $D = \emptyset$ at step i, and hence there is no execution with signature $C_0^S \dots C_i^S$ that deadlocks at C_i, the randomly chosen execution is not a witness for deadlock. If the answer is true, continue

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

Let us assume that the universal termination question is considered for a single input dataset D_{in}, one has to check that $D_{in} \models \psi_0$. As ψ_0 is in universal i.e. is of the from $\forall \vec{X} \phi_0$, this can be done in $O(|D_{in}|^{|\psi_0|}$. If the answer is true then we have found preconditions that are satisfied by D_{in} and that are sufficient to obtain an empty dataset at configuration C_i in a run $C_0 \dots C_i$ that has signature $C_0^S \dots C_i^S$, i.e. $C_0^S \dots C_i^S$ witnesses the existence of a deadlock. Overall, one has to solve up to $i < 3.K_{\mathcal{T}_{cx}}$ satisfiability problems for universal FO formulas $\psi_{i-1}, \psi_{i-2} \dots \psi_1$ of size smaller than r^i, and a model checking problem for input D_{in} with a cost in $O(|D_{in}|^{r^i})$. The satisfiability problems are NEXPTIME in the size of the formula [20], hence

a complexity that is doubly exponential in $K_{\mathcal{T}_{cx}}$. Considering that data fields are encoded with c bits of information, D_{in} is a dataset of size in $O(2^{r.c})$. Hence, the overall complexity to check that $C_0^S \dots C_i^S$ is a witness path that deadlocks in $2 - EXPTIME$.

Conversely, if the universal termination question is considered for a several input datasets described with a FO formula $\mathcal{D}_{in}$, one has to check that no contradiction arises when requiring existence of an input D_{in} that satisfies both $\mathcal{D}_{in}$ and ψ_0. This can be done by checking the conjunction $\mathcal{D}_{in} \wedge \psi_0$. Now, the formula is of size $|\mathcal{D}_{in}| + |\psi_0|$, and more importantly falls in the class of $\mathcal{D}_{in}$. Hence, this last step is a combination of an NP-problem over a formula of size in $O(r^i)$, and of a satisfiability problem which complexity is in $O(2^{|\mathcal{D}_{in}|})$ for universal FO formula, doubly exponential in $|\mathcal{D}_{in}|$ for BSR. Letting n_{in} denote the number of variables in $\mathcal{D}_{in}$, the complexity of the last step is n_{in}-fold exponential in the size of $\mathcal{D}_{in}$ for separated formulas. As for the unique input case, if the answer is true, then $C_0^S \dots C_i^S$ witnesses existence of a non-terminating execution.

Hence, one can witness existence of a non-terminating run in $O(K_{\mathcal{T}_{cx}}.2^{r^{K_{\mathcal{T}_{cx}}}} + C_{in})$ where C_{in} is the cost required to check satisfiability of the constraint on inputs, that is in $co - 2EXPTIME$ (in $K_{\mathcal{T}_{cx}}$) if $\mathcal{D}_{in}$ in the universal fragment, or in the BSR fragment (the complexity of checking $\psi_i \dots \psi_0$ and $\mathcal{D}_{in}$ is in $2 - EXPTIME$), and $co - n - foldEXPTIME$ for the BSR fragment.

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

One can notice that the algorithms to check universal termination of CWs are non-deterministic, and that in the worst case, one may have to explore all symbolic executions of size at most $3 \cdot K_{\mathcal{T}_{cx}}$. All these executions can be grouped in a common structure, i.e. a *symbolic execution tree* representing possible sequences of moves starting from the initial configuration.

Definition 14 (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 symbolic configurations of the form $C_i^S = (W_i, Ass_i, \mathbf{Dass}^S)$, where $W_i = (N_i, \longrightarrow_i, \lambda_i)$ and $Ass_i : N_i \to \mathcal{U}$ are the usual workflow and worker assignment relations, and $\mathbf{Dass}^S$ associates a sequence of relational schemas to minimal nodes of W_i. $(C_i^S, C_j^S) \in E$ if $C_j^S \in SC(C_i^S)$.*

Each path of the tree defines a symbolic execution. The symbolic execution tree of a complex workflow is a priori an infinite structure, but for recursion free CWs, the tree is of bounded depth, and bounded degree. One can hence perform an exhaustive search for deadlocks and potential deadlocks in the symbolic execution tree of recursion free workflow to exhibit a witness for non-termination. Let $C_i^S = (W_i, Ass_i, \mathbf{Dass}_i)$ be a potential deadlock symbolic configuration. 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 to a worker and have several successors in W_i. Let $\Pi = C_0^S \dots C_i^S$ be the path from the root of the tree to a potential deadlock C_i^S. Even if vertex C_i^S has a successor C_k^S, obtained by executing a 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 Π. Hence, some of the tasks execution moves depicted in $\mathcal{B}$ may not be realizable. If executing task $\lambda(n_j)$ is the only possible action from C_i^S and if a run with signature Π 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 Π end with data assignments that affect non-empty sequences of datasets to all nodes of S, then C_i^S will never cause a real deadlock. Note also that when C_i^S is a potential deadlock, there exists necessarily a path $C_i^S.C_{i+1}^S \dots C_{i+h}^S$ in $\mathcal{B}$ where the only actions allowed from C_{i+h}^S is the execution of the incriminated minimal tasks that needs non-empty inputs.

First of all, assume that a path in $\mathcal{B}$ exists from C_0^S to some vertex C_k^S where a dataset D_k has to be split. As explained in the proof of Theorem 2, the property that $D_k = \emptyset$ is expressible in the universal fragment of FO, so one can check satisfiability of a sequence of weakest preconditions up to ψ_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 executions, 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 $C_0^S \dots C_{dead}^S$ to a deadlock vertex C_{dead}^S exists after verifying that potential deadlocks

Algorithm 1: Universal Termination Decision

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

If CW has unbounded recursion **Return** NO-TERM
Build the symbolic execution tree $\mathcal{B} = (V, E)$ of CW
$V_{split} = \{(W, Ass, \mathbf{Dass}^S) \in V \mid W \text{ has splittable nodes}\}$
for $v \in V_{split}$ **do**
  $\rho_v = C_0 \xrightarrow{a_0} C_1 \ldots \xrightarrow{a_{k-1}} C_k = v$ `//path from` v_0 `to` v
  for $n \in$ *split nodes of* $min(W_i)$ **do**
    $WP ::= wp[a_{k-1}](\bigvee_{D_k \in \mathbf{Dass}^S(n)} D_k = \emptyset)$
    for $i = k - 1..1$ **do**
      Check satisfiability of WP
      if WP *not satisfiable* **then**
        break; //unfeasible path
      end
      $WP ::= wp[a_{i-1}]WP$
    end
    `//WP= WP0`
    if WP *satisfiable* $\wedge D_{in} \models WP$ **then**
      return *NO-TERM*
    end
  end
end
`//All Split nodes have non-empty input datasets`
if $\exists v \in V$ *without successors and* v *is not final* **then**
  return *NO-TERM*
end
return *TERM*

are harmless, then a run with signature $C_0^S \ldots C_{dead}$ 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 $\mathcal{D}_{in}$.

6.3 Termination with a guaranteed bound

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 [7] (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 1, 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. Obviously, recursion-free CWs cannot encode counter machines, and have a bounded number of signatures, as their runs are of length at most $3.K_{\mathcal{T}_{cx}}$. This immediately gives us the following corollary:

Corollary 2. *Let CW be a recursion-free complex workflow, which runs length is bounded by $3.K_{\mathcal{T}_{cx}}$. Let D_{in} be a dataset, and $\mathcal{D}_{in}$ an FO formula. One can decide in $2 - EXPTIME$ (in $K_{\mathcal{T}_{cx}}$) whether CW terminates existentially on input D_{in}. If $\mathcal{D}_{in}$ is in the existential, universal or BSR fragment of FO, then existential termination of CW is also in $2EXPTIME$. As it is sufficient to exhibit non-deterministically terminating run of size at most $3.K_{\mathcal{T}_{cx}}$.*

Proof. The proof follows the same line as for Theorem 2: one has to find non-deterministically a signature of size at most $3.K_{\mathcal{T}_{cx}}$. When such a signature ρ^S is found it remains to compute $|\rho^S|$ weakest preconditions $\psi_{|\rho^S|}, \ldots \psi_0$, imposing at each step that inputs of automated or split tasks are not empty. After this verification, it remains to show that inputs satisfy ψ_0. □

Existential termination is hence decidable for recursion-free CWs, provided the description of inputs belongs to a decidable fragment of FO. One can notice that the decision procedure is doubly exponential in the length of runs (that is in $K_{\mathcal{T}_{cx}}$). This bound can itself be exponential (see proof of prop. 4), but in practice, one can expect refinements to stop only in a few steps, as refinements are supposed to transform a complex task into an orchestration of simpler subtasks). With this assumption, $K_{\mathcal{T}_{cx}}$ remains a simple polynomial in the number of complex tasks. Another way to bound recursiveness in a complex workflow is to limit the number of refinements that can occur during an execution. We can slightly adapt the semantics of Section 4, and in particular rule R_4, and replace it by a *restrictive decomposition* (*RD*) rule. Intuitively, the (*RD*) 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 $T = \{t_{int}, t_2, ..., t_f\}$ be a set of tasks of size n. Let $KD = (k_1, k_2, ..., k_n) \in \mathbb{N}^n$ be a vector constraining the number of refinements of task t_i that can occur in a run ρ. 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:

$$\frac{\begin{array}{l}\exists n \in min(W), \exists u = Ass(n), t_i = \lambda(n) \in T_{cx} \wedge S[i] \leq k_i \\ \wedge \exists W_s = (N_s, \longrightarrow_s, \lambda_s) \in Profile(t_i, u) \\ \wedge Ass' = Ass \setminus \{(n, Ass(n))\} \wedge \mathbf{Dass}'(min(W_s)) = \mathbf{Dass}(n) \\ \wedge \forall x \in N_s \setminus min(W_s), \mathbf{Dass}'(x) = \emptyset^{|Pred(x)|} \\ \wedge W' = W_{[n/W_s]} \\ \forall j \in 1 \ldots |T|, S'[j] = S[j] + 1 \text{ if } j = i, S[j] \text{ otherwise}\end{array}}{(W, Ass, \mathbf{Dass}, S) \xrightarrow{split(t_i)} (W', Ass', \mathbf{Dass}, S')} \quad (5)$$

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 8. *Let $\rho = C_0 \ldots C_q$ be a run of a complex workflow under RD semantics allowing at most k refinements of each task. The length of ρ is bounded by $L(n, k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$*

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

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 $\Pi = V_0 \ldots V_n$. 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 Π 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, with the same complexity as for recursion-free specifications.

Theorem 3. *Let CW be a complex workflow with restricted decomposition semantics. Then:*

- *Existential termination of CW for a single input D_{in} or for a set of inputs $\mathcal{D}_{in}$ described with the universal, existential, or BSR fragment of FO is in 2EXPTIME.*
- *Universal termination of CW for a single input D_{in} or for a set of inputs $\mathcal{D}_{in}$ described with the universal, existential, or BSR fragment of FO is in co − 2EXPTIME.*

Proof. Under restricted decomposition semantics, the length of a run is bounded by $3 \cdot k \cdot \mathcal{T}_{cx}^2 + 3 \cdot |W_0|$. We can reuse the techniques of Theorem 2 to find a witness symbolic run that is the signature of a run that does not terminate, and of corollary 2 to find a witness symbolic run that is the signature of a run that terminates. □

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

- *$\psi_E^{in,out}$ is a conjunction of existential formulas addressing the contents of the input/output dataset, i.e. constraints of the from $\exists x, y, z, rn(x, y, z) \in D_{in} \wedge P(x, y, z)$,*
- *$\psi_A^{in,out}$ is a conjunction of universal formulas constraining all tuples of the input/output dataset, of the form $\forall x, y, z, rn(x, y, z) \in D_{in} \Rightarrow P(x, y, z)$ where P is a predicate on values of x, y, z.*
- *$\psi_{AE}^{in,out}$ is a conjunction of formulas relating the contents of inputs and outputs, of the form $\forall x, y, z, rn(x, y, z) \in D_{in} \Rightarrow \exists (u, v, t), \phi(x, y, z, u, v, t)$*

Definition 16 (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 $\mathcal{R}uns(CW, D_{in})$* 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 $\mathcal{R}uns(CW, D_{in})$ terminate properly for every $D_{in} \in \mathcal{D}_{in}$*

Proper termination considers correctness of the computed outputs in addition to termination of runs. In general, termination of a run does not guarantee its proper termination. A terminated run starting from an input dataset D_{in} may return a dataset D_{out} such that pair D_{in}, D_{out} does not comply with 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. Obviously, a CW that does not terminate cannot terminate properly, and setting $\psi^{in,out} ::= true$ recasts the proper termination question into Proper termination can be immediately brought back to a termination question, by setting $\psi_{in,out} = true$. We hence have the following corollary.

Corollary 3. *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 4. *Let CW be a complex workflow, and $\psi^{in,out}$ be a constraint. Then:*

- *existential and universal proper termination of CW are undecidable, even under RD semantics.*
- *if $\psi^{in,out}$ is in a decidable 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, and in particular for the AE fragment with formulas of the form $\forall \vec{X} \exists \vec{Y}, \phi(X, Y)$. Hence $\psi^{in,out}_{AE}$ can be a formula which satisfiability is not decidable. One can take an example of formula ψ_{unsat} which satisfiability is not decidable. One can also build a formula ψ_{id} that says that the input and output of a workflow are the same. One can design a workflow CW_{id} with a single final node which role is to return the input data, and set as client constraint $\psi^{in,out} = \psi_{unsat} \wedge \psi_{id}$. This workflow has a single run, 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 ψ_{unsat} is satisfiable. Universal and existential proper termination are hence undecidable problems.

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

Let us first consider universal termination. For this, one can compute a chain of weakest preconditions $\psi_n, \psi_{n-1}, \dots \psi_0$ that have to be enforced to execute successfully CW and terminate in node n. In particular, $\psi_n ::= true$. Similarly, one can compute at each step, a weakest precondition $\psi^{in,out}_i$ needed at step i so that $\psi^{in,out}$ holds. If at one stage, ψ_i or $\psi^{in,out}_i$ is not satisfiable, then ρ^S is not the signature of an actual run of CW that terminates properly, and we have found a witness of non-proper termination. We have assumed that $\psi^{in,out}$ was specified in a decidable fragment of FO. As computing the weakest precondition of a property in the existential, universal, BSR, SF fragment of FO gives a property in the same fragment, all ψ_i's and $\psi^{in,out}_i$'s are in a decidable fragment of FO. Then, the complexity will depend on the considered fragment, and on the fragment of FO used to specify inputs. As for universal termination, if inputs and $\psi^{in,out}$ are specified with the universal and existential fragments, then universal proper termination is in $co - 2EXPTIME$. If $\psi^{in,out}$ is in SF, then checking proper universal termination is $co - K - fold-$exponential time, where $K = r^{K\tau_{cx}}$.

Then, one has to check satisfiability of $\psi_{proper,n} ::= \bigwedge \psi_i \wedge \psi^{in,out}$. So, if $\psi^{in,out}$ is written in a decidable fragment, 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 a decidable fragment of FO.

The prooof and complexities for existential termination follow the same lines, yielding $2EXPTIME$ complexity when $\psi^{in,out}$ is written with the existential, universal, fragments of FO, $3 - EXPTIME$ complexity for when $\psi^{in,out}$ is written in the BSR fragment (as checking a BSR formula in in NEXPTIME []) and $K - fold-$exponential for SF formulas. □

At first sight, restricting to the existential, universal, BSR, or SF fragments 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

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

Formula $\psi^{valid}_{in,out}$ 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.

Last, one can also consider formulas in which $\psi^{in,out}_{EA}$ is of the form $\forall x_1, \dots x_k \exists y_1, \dots, y_q \phi$ as soon as every atom in ϕ that is not separated contains only existential variables that take values from a finite domain. Then $\psi^{in,out}_{AE}$ can be transformed into an equivalent universal formula which matrix is a boolean expression on separated atoms

8 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 questions remain open: satisfiability of separated FO is n-fold exponential [30], which makes the SF fragment a priori unusable. However, preconditions used in the termination decision procedures are all existential or universal formulas, and SF formulas only originate from input descriptions. Hence, the complexity of termination should be in $2EXPTIME$ for most cases. One can should also consider that the length of runs to consider is a priori exponential, but that a reasonable crowdsourcing platform will probably not give too much refinement power to its workers. We can hence expect $K_{\mathcal{T}_{cx}}$ to be rather small. So far, we do not know whether the complexity bounds are sharp. 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 tradeofs between performance and accuracy of answers...

References

1. S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active XML systems. *Trans. Database Syst.*, 34(4):23:1–23:44, 2009.
2. S. Abiteboul and V. Vianu. Collaborative data-driven workflows: think global, act local. In *Proc. of PODS'13*, pages 91–102. ACM, 2013.
3. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for Web services (BPEL4WS). version 1.1, 2003.
4. Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying business processes. In *Proc. of VLDB'06*, pages 343–354. ACM, 2006.
5. P. Bernays and M. Schönfinkel. Zum entscheidungsproblem der mathematischen logik. *Mathematische Annalen*, 99(1):342–372, 1928.
6. E. F. Codd. Relational completeness of data base sublanguages. *Database Systems*, pages 65–98, 1972.
7. E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *Trans. on Database Systems*, 37(3):22, 2012.

8. F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh. Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. *ACM Computing Surveys*, 51(1):7, 2018.
9. A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proc. of PODS'06*, pages 90–99. ACM, 2006.
10. E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of program. *Communications of the ACM*, 18(8):453–457, 1975.
11. H. Garcia-Molina, M. Joglekar, A. Marcus, A. Parameswaran, and V. Verroios. Challenges in data crowdsourcing. *Trans. on Knowledge and Data Engineering*, 28(4):901–911, 2016.
12. Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *Proc. of PODS 2013*, pages 163–174. ACM, 2013.
13. S. Itzhaky, T. Kotek, N. Rinetzky, M. Sagiv, O. Tamir, H. Veith, and F. Zuleger. On the automated verification of web applications with embedded SQL. In *Proc. of ICDT'17*, volume 68 of *LIPIcs*, pages 16:1–16:18, 2017.
14. L.G. Khashiyan. Polynomial algorithms in linear programming. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 20:51–68, 1980.
15. D. Kitchin, W.R. Cook, and J. Misra. A language for task orchestration and its semantic properties. In *CONCUR'06*, pages 477–491, 2006.
16. A. Kittur, B. Smus, S. Khamkar, and R.E. Kraut. Crowdforge: Crowdsourcing complex work. In *Proc. of UIST'11*, pages 43–52. ACM, 2011.
17. A. Koutsos and V. Vianu. Process-centric views of data-driven business artifacts. *Journal of Computer and System Sciences*, 86:82–107, 2017.
18. P. Kucherbaev, F. Daniel, S. Tranquillini, and M. Marchese. Crowdsourcing processes: A survey of approaches and opportunities. *IEEE Internet Computing*, 20(2):50–56, 2016.
19. A. Kulkarni, M. Can, and B. Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *Proc. of CSCW'12*, pages 1003–1012. ACM, 2012.
20. Harry R. Lewis. Complexity results for classes of quantificational formulas. *J. Comput. Syst. Sci.*, 21(3):317–353, 1980.
21. G. Li, J. Wang, Y. Zheng, and M.J. Franklin. Crowdsourced data management: A survey. *Trans. on Knowledge and Data Engineering*, 28(9):2296–2319, 2016.
22. G. Little, L.B. Chilton, M. Goldman, and R.C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *Proc. of HCOMP'09*, pages 29–30. ACM, 2009.
23. L. Löwenheim. Über möglichkeiten im relativkalkül. *Mathematische Annalen*, 76(4):447–470, 1915.
24. P. Mavridis, D. Gross-Amblard, and Z. Miklós. Using hierarchical skills for optimized task assignment in knowledge-intensive crowdsourcing. In *Proc. of WWW'16*, pages 843–853. ACM, 2016.
25. J. Misra and W. Cook. Computation orchestration. *Software and Systems Modeling*, 6(1):83–110, 2007.
26. M. Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.
27. A. Nigam and N.S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
28. OASIS. Web Services Business Process Execution Language. Technical report, OASIS, 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.
29. D. Sánchez-Charles, V. Muntés-Mulero, M. Solé, and J. Nin. Crowdwon: A modelling language for crowd processes based on workflow nets. In *AAAI*, pages 1284–1290, 2015.
30. T. Sturm, M. Voigt, and C. Weidenbach. Deciding first-order satisfiability when universal and existential variables are separated. In *Proc. of LICS '16*, pages 86–95. ACM, 2016.
31. Stefano Tranquillini, Florian Daniel, Pavel Kucherbaev, and Fabio Casati. Modeling, enacting, and integrating custom crowdsourcing processes. *TWEB*, 9(2):7:1–7:43, 2015.
32. W.M.P Van Der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, HMW Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.
33. Q. Zheng, W. Wang, Y. Yu, M. Pan, and X. Shi. Crowdsourcing complex task automatically by workflow technology. In *MiPAC'16 Workshop*, pages 17–30. Springer, 2016.

A Appendix : proofs

A.1 Proof of Theorem 1

Theorem 1 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 $inst_q = inc(q, c_l, q')$, depicting the fact that the machine is in state q, increases the value of counter c_l by 1, and moves to a new state q'. Instructions in I_2 are of the form $inst_q = dec(q, c_l, q', q'')$, depicting the fact that the machine is in state q, if $c_l == 0$, the machine moves to new state q' without making any change in the value of counter c_l, and otherwise, decrements the counter c_l and moves to state q''. We consider deterministic machines, i.e. there is at most one instruction $inst_q$ per state in $I_1 \cup I_2$. At any instant, the machine is in a configuration $C = (q, v_1, v_2)$ where q is the current state, v_1 the value of counter c_1 and v_2 the value of counter c_2.

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 $\rho = C_0.C_I \dots C_k$ 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 $(n_q^{inc}, n_{q'}) \in \longrightarrow$, $\lambda(n_q^{inc}) = t_q^{inc}$ and $\lambda(n_{q'}) = t_{q'}$. Task t_q^{inc} is an atomic task that adds one record of the form (k', Cnt_x) to the dataset. Hence, after executing tasks t_q and t_q^{inc}, the number of occurrences of Cnt_x has increased by one.
- For every instruction of the form $dec(q, c_x, q', q'')$, we create a complex task t_q and a worker u_q who can choose to refine t_q according to 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 $n_q^{div}, n_q^{C_x}, n_q^{C_{\bar{x}} \cup \perp}, n_q^{\otimes}, n_q^{dec}$ and $n_{q'}$, respectively labeled by tasks $t_q^{div}, t_q^{C_x}, t_q^{C_{\bar{x}} \cup \perp}, t_q^{\otimes}, t_q^{dec}$ and $t_{q'}$. The dependence relation in $W_{q,NZ}$ contains pairs $(n_q^{div}, n_q^{C_x}), (n_q^{div}, n_q^{C_{\bar{x}} \cup \perp}), (n_q^{C_x}, n_q^{\otimes}), (n_q^{C_{\bar{x}} \cup \perp}, n_q^{\otimes})$, $(n_q^{\otimes}, n_q^{dec})$ and $(n_q^{dec}, n_{q'})$. The role of t_q^{div} is to split $\mathbf{Dass}(n_q^{div})$ into disjoint parts: the first one contains records of the form $R(k, C_x)$ and the second part consists of all other remaining records. Tasks $t_q^{C_x}$ and $t_q^{C_{\bar{x}} \cup \perp}$ simply forward their inputs, and task $t_q^{\otimes}$ computes the union of its inputs. Note however that if one of the inputs is empty, the task cannot be executed. Then, task t_q^{dec} deletes one record of the form $R(k, C_x)$. Hence, if $D_q = \mathbf{Dass}(n_q)$ is a dataset that contains at least one record of the from $R(k, C_x)$, the execution of all tasks in $W_{q,NZ}$ leaves the system in a configuration with a minimal node $n_{q'}$ labeled by task $t_{q'}$, and with $\mathbf{Dass}(n_{q'}) = D_q \backslash R(k, C_x)$
- Let us now detail the contents of $W_{q,Z}$. This workflow is composed of nodes $n_q^{div}, n_q^{C_x \cup \perp}, n_q^{id}, n_q^{btest}, n_q^{done}, n_{q'}$ respectively labeled by tasks $t_q^{div}, t_q^{C_x \cup \perp}, t_q^{id}, t_q^{btest}, t_q^{done}, t'_q$. The flow relation if given by pairs $(n_q^{div}, n_q^{C_x \cup \perp})$, (n_q^{div}, n_q^{id}) $(n_q^{C_x \cup \perp}, n_q^{btest})(n_q^{btest}, n_q^{done})$ and (n_q^{id}, n_q^{done}). The role of task t_q^{div} is to project its input

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

One can see that for every run $\rho = C_0 \dots C_k$ of the two counter machine, 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.

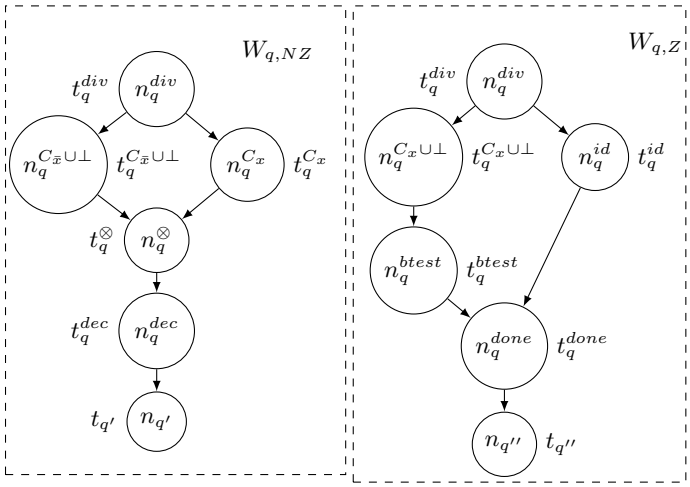

Fig. 9. Encoding of Non-zero test followed by decrement (left), and Zero Test followed by state change (right).

A.2 Proof of Proposition 7

Lemma 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 Π 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 Π.

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 7: 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 ρ_ω, 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 ρ_ω 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 2

Theorem 2 : 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 ρ_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 $RG = (\mathcal{T}, \longrightarrow_{\mathcal{T}}, T_0)$ where T_0 is the set of tasks that appear in W_0, $(t, t') \in \longrightarrow_{\mathcal{T}}$ iff there exists a worker u, a workflow $W_t = (N_t, \longrightarrow_t, \lambda_t)$ 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 $\nexists x_1, \dots x_k, rn(x_1, \dots x_k) \in D$. 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 2.

A.4 Proof of Proposition 8

Proposition 9. *Let $\rho = C_0.C_1 \dots C_q$ be a run under RD semantics. Then, for every $C_i = (W_i, Ass_i, \mathbf{Dass}_i)$, the number of nodes in W_i is smaller than $C(n,k) = k.n^2 + |W_0|$.*

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 ρ 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 ρ 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 $k.n$ times in a run following the RD semantics, even if this run is of length greater than $k.n$. 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 $S_{k.n} = |W_0| + (k.n)(n-1) = |W_0| + k \cdot n^2 - k \cdot n$.

Proposition 8: Let $\rho = C_0 \dots C_k$ be a run of a complex workflow under *RD* semantics. The length of ρ is bounded by $L(n,k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$

Proof. Recall that a configuration is a triple $C_i = (W_i, Ass_i, \mathbf{Dass}_i)$, 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 **Dass** 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 ρ 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, $|\rho|_1 = |\rho|_2 + |\rho|_4$. 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 ρ is bounded by $L(n,k) = 3 \cdot k \cdot n^2 + 3 \cdot |W_0|$

A.5 Proof of Proposition ??

Proposition ??: Let ψ_i be an FO formula. Then, for any move m_i from C_{i-1} to C_i and any formula ψ_i, $\psi_{i-1} = wp[m_i]\psi_i$ is an effectively computable FO Formula. Further, if ψ_i is in separated form, then ψ_{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 write $D_1, \dots D_k \models \psi$ to denote that a set of datasets satisfies formula ψ. 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 is 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 $DU \models \exists w_1, \dots w_p, nb\ rn_u(w_1 \dots w_p, nb) \wedge (nb = 3) \wedge \phi$.

Given a transformation tr that transforms inputs $D_1, \dots, D_n$ into outputs $D'_1, \dots, D'_k$, and an FO property ψ_{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 ψ_{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 ψ_{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 ψ_{pre} is the precondition on datasets in use in C required for ψ_{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 ψ_{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 ψ, we denote by $Vars(\psi)$ the variables $x_1 \dots x_n$ used in ψ. 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 $\forall, \exists$), ϕ 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 ψ. We 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 2 (Weakest precondition for Projection). *Let ϕ 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 ϕ is a separated FO formula, then ψ 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 ψ 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 ψ and is obtained by projection of a record $r' = (v'_1, \dots, v'_k)$ with relational schema rs_i, then r' also satisfies ψ. Similar reasoning holds for 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'_is are new variables addressing values of fields in $A_i \setminus A_j$. We denote by $\psi_{post_{[rs_j/rs_i]}}$ the formula ψ_{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 $D'_1, \dots D_i, \dots, D'_k$, 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 ψ_{post} is separated, $wp[Proj]\psi_{post}$ is also separated.

Lemma 3 (Weakest precondition for R2R transformations). *Let ϕ 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 ϕ is a separated FO formula, then ψ 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 ψ_{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 $w_j = k_{j,1}v_1 + k_{j,2}v_2 + k_{j,p}v_p + k_j$. 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 ψ_{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 ψ_{post} can address values of more that 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 ψ_{post}. We denote by $\psi_{post_{[B/R2R(A)]}}$ the formula ψ_{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 ψ_{post} is separated, then $wp[R2R]\psi_{post}$ is also separated.

Lemma 4 (Weakest precondition for Selection of records). *Let ϕ 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 ϕ and P are separated FO formulas, then ψ 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 ψ_{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 ϕ addressing tuples with relational schemas in rs. For $i \in 1..K_{rn}$, we let $\phi^P_{rn,i}$ denote the formula $rn(w_i, \dots w_{i+k}) \wedge P(w_i, \dots w_{i+k})$ if $\phi_{rn,i}$ is in positive form and $\neg(rn(w_i, \dots w_{i+k}) \wedge P(w_i, \dots w_{i+k}))$ otherwise. Last, let us denote by $\phi_{[\{\phi_{rn,i}\}|\{\phi^P_{rn,i}\}]}$ 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

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

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

Lemma 5 (Weakest precondition for field addition). *Let ϕ 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 ϕ is a separated FO formulas, then ψ 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. 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 sub-formulas of the form $rn_j(...)$ or $\neg rn(...)$ in ψ_{post} (again these sub-formulas are over disjoint variables). Formula ψ_{post} is hence a formula over variables $W = Vars(\phi_{post})$ that contain at least a set of variables $w_1, \dots w_p, w_{p+1} \dots w_{K_{fld}.p}$ appearing in relational sub-formulas. ψ_{post} is of the form $Q(Vars(\psi_{post})), \phi$, where ϕ is a boolean combination of relational statements and comparisons of field values. Here, we can transform ϕ 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 sub-formula $rn_i(w_k, \dots w_{p+k-1}) \wedge P_{add}(w_k, \dots w_{p+k})$ in positive sub-formulas, and by a sub-formula 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 ψ_{post} is separated. Then, as for the 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.

Lemma 6 (Weakest precondition for field enlargement). *Let ϕ 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 ϕ is a separated FO formula, then ψ 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 δ. 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 $r_2 = (v_1, \dots v'_j, \dots, w_p)$ such that $v_j \in [v_j - \delta, v_j + \delta] \cap Dom(a_j)$. One can notice that enlargement preserves the relational schema of input dataset D_i.

Let ψ_{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 ψ_{post} as a formula over $V = W \cup Y$ where $W = w_1, \dots w_p, w_{p+1} \dots w_{K_{rn}.p}$ is the set of variables used in these relational statements and Y the other variables. ψ_{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 $W_{imp} = w_j, w_{j+p}, \dots w_{(K_{rn}-1).p+j}$

ψ_{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 $\phi^{imp} = expr_{1_{[v_{j+n.p}/v_{j+n.p}+x_{j+n.p}]}} \bowtie expr_{2_{[v_{j+n.p}/v_{j+n.p}+x_{j+n.p}]}}$ 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 from $\phi = rn(v_1, \dots v_j, \dots v_k)$ 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 ϕ_1, ϕ_2, ϕ^{imp} is the formula obtained as a boolean combination of ϕ_1^{imp} and ϕ_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 ϕ. We now define the weakest precondition for ψ_{post} that has to be satisfied by $D'_1, \dots D_i, \dots D'_k$ when enlarging field a_j as

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

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

Lemma 7 (Weakest precondition for unions of datasets). *Let ϕ 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 ϕ is a separated FO formula, then ψ 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, ψ_{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 ψ_{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 ψ_{post} holds for dataset D_a iff one can build a map $aff : 1..KU \to 1..q$ 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 ψ_{post}. For a particular assignment $aff \in Assign(KU, q)$ we can write a formula ϕ^{aff}_{post} 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 $D'_1, \dots, D_1, \dots D_q, \dots D'_k \models \phi^{aff}_{post}$. 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 ψ_{post} hold on $D'_1, \dots D_a, \dots D'_k$ after merging $D_1, \dots D_q$ is

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

As variables used in atoms of $wp[Union]\psi_{post}$ do not change with respect to the original formula, if ψ_{post} is separated, then all atoms in $wp[Union]\psi_{post}$ also separated universally and existentially quantified variables.

Lemma 8 (Weakest precondition for joins). *Let ϕ 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 ϕ and P are separated FO formulas, then ψ 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 $rs_2 = (rn_2, A_2)$ 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 ψ_{post}, the weakest precondition that has to be computed is a property of $D_1, D_2, D'_3, \dots D'_k$.

Formula ψ_{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 ψ_{post} is of the form $Q(V), \phi_{post}$. Now, every positive statement of the from $rn_a(w_i, w_{i+p_1+p_2-1})$ 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 $Vars(\psi_{post}$. Let $\psi_{post_{[D_a|D1,D2]}}$ 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 $\psi^{prenex}_{post_{[}D_a|D1,D2]}$ 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 ψ_{post} to hold after a join is:

$$wp[Join]\psi_{post} = \psi^{prenex}_{post_{[}D_a|D1,D2]}$$

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

Lemma 9 (Weakest precondition for record insertion). *Let ϕ be a FO formula, and act be an atomic action that inserts a new record is a dataset. Then one can effectively compute an FO formula $\psi = wp[act]\phi$. Moreover, —if ϕ is a separated FO formulas, then ψ 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 ψ_{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 ψ_{post} be of the form $Q(V), \phi$, with $V = W \cup Y$, and W be the variables appearing in relational clauses of the from $rn(w_i, \dots w_{i+p})$. Let us assume that ϕ is a quantifier free formula in disjunctive normal form. In other words, ϕ is of the from $\phi = \bigvee_{k=1..K} \phi_k = at_{k,1}(V) \wedge \dots \wedge at_{k,m_k}(V)$, 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 $\exists \overrightarrow{V}_1, \forall \overrightarrow{V}_2 \phi'_k$. 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, ϕ'_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 $rn(..) \in D'_j$. 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 $P(w_i, \dots w_{i+p}, x, y, z, ...)$ 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 $x, y, z....$ 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 ϕ_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 ψ_{post} to hold after insertion, there must be at least a correct assignment.

The precondition for ψ_{post} that has to be satisfied by $D'_1, \dots D_i \dots D'_k$ hence becomes : $wp[Additive]\psi_{post} = \bigvee_{k=1..K} \bigvee_{1..2^{AK}} \exists V_1 \setminus V_X \forall V_2 \phi_{k \setminus Ass_X}$

One can notice that if ψ_{post} is separated, the resulting formula is still separated.

Lemma 10 (Weakest precondition for record deletion). *Let ϕ 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 ϕ is a separated FO formulas, then ψ 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 ψ_{post}, or r is a record that falsifies ψ_{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 ψ 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 $\not\psi$. The weakest precondition for deletion that has to be satisfied by $D'_1, \dots D_i \dots D'_k$ becomes : $wp[Del]\psi = Q(V), \psi \wedge \bigwedge_{X \in Ass} Q(V \setminus) \neg\psi_X$

Notice that if ψ is in separated from, then $wp[Del]\psi$ is also in separated form.

Lemma 11 (Weakest precondition for datasets decomposition). *Let ϕ 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 ϕ is a separated FO formula, then ψ is also separated.*

Proof. The **Decomposition** of a task is a higher order operation to split a task into several orchestrated subtasks. 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 ψ is of the form $\psi = Q(V), \phi$, and such that ϕ contains positive relational statement of the form $rn_j(w_1..w_{|A_j|}) \in D'_j$, and negative relational statements of the from $\neg rn_j(w_1..w_{|A_j|}) \in D'_j$. As we have that D'_j is a dataset obtained as a function $f_j(D_i)$ we can rewrite ψ as an equivalent formula $\psi_2 = Q(V), \phi_2$, where ϕ_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 ϕ. 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 $r = (v_1, \ldots v_p)$ in D_i that is a solution for formula $P_j(w_i, \ldots w_{i+p})$. Equivalently, a positive statement of the from $rn_j(w_k, \ldots w_{k+|A_j|-1}) \in f_j(D_i)$ can be replaced by $rn_j(v_k, \ldots v_{k+|A_i|-1}) \in D_i \wedge P_j(w_k, \ldots w_{k+|A_j|-1})$, and a negative statement of the form $\neg rn_j(v_k, \ldots v_{k+|A_j|-1}) \in f_j(D_i)$ can be replaced by $\neg \big(rn_j(v_k, \ldots v_{k+|A_i|-1}) \in D_i \wedge P_j(w_k, \ldots w_{k+|A_j|-1})\big)$. Letting ϕ_3 be the formula ϕ_2 where every relational statement has been replaced this way, and letting $Q'(V)$ denote the prefix in which every instance of $w_k, \ldots w_{k+|A_j|-1}$ is replaced by fresh variables $v_k, \ldots v_{k+|A_j|-1}$, the weakest precondition needed such that $D'_1, \ldots D_1, \ldots D_l \ldots D'_k \models \psi$ 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 $v_k, \ldots v_{k+|A_i|-1}$ then one can also replace
$rn_j(w_k, \ldots w_{k+|A_j|-1})$ 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

$$\begin{aligned} D_i \models \exists v_1, v_2, z, t,\, & rn_1(v_1, v_2) \in D_i \wedge v_1 < 10 \\ & \wedge rn_2(v_3, v_4) \in D_i \wedge v_3 < v_4 \wedge v_2 = v_3 \end{aligned}$$

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

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 $\rho = C_0 \ldots C_n$ 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 $WP_0 \ldots WP_{n-1}$ 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 $WP_i = D'_1, \ldots D'_m \models \psi$ 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 $C_{i-1} \longrightarrow C_i$, WP_{i-1} is of the form $D'_1, \ldots D'_q \models \psi_{i-1}$, where ψ_{i-1} is computed inductively as $wp[op_1](wp[op_2](\ldots wp[op_k]\psi_i))$, and $op_1, \ldots op_k$ is the sequence of operations used to transform datasets D_1, D_q in C_{i-1} into $D'_1, \ldots D'_m$ in C_i. The weakest precondition for move from C_{i-1} to C_i is hence $D'_1, \ldots D'_m \models wp[op_1](wp[op_2](\ldots wp[op_k]\psi_i))$.

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 $f_1, \ldots f_k$, to obtain new datasets and new data assignments. If WP_i is of the form $D_1, \ldots D_m$ and datasets $D_n, \ldots D_{n+k}$ 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 $D_1, \ldots D'_j, \ldots D_m \models \wp[add/remove]\psi_i$. 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 ρ 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 ψ 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 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 ρ. 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 $\theta(m^{2^k})$. 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 ρ. Hence, in the worst case the total number of new linear inequalities becomes $\sum_{i=1}^{n} m_i^{2^{k_i}}$. 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.

Lemma 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 ρ. 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 ψ_j is a separated FO formula.*

Proof. The proof follows from the lemma **??**. In a run ρ, 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 $v_n \to v_{n-1} \to v_0$. 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 **??**, 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 ψ_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 ψ_{post_j}. Hence, for every move m_j there exist a weakest precondition WP_j such that $D_1, \dots D_{m_j} \models \psi_j$.

With all the above lemmas, we have shown that the weakest precondition for a 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.

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 with respect to their data assignment, and E represents moves among

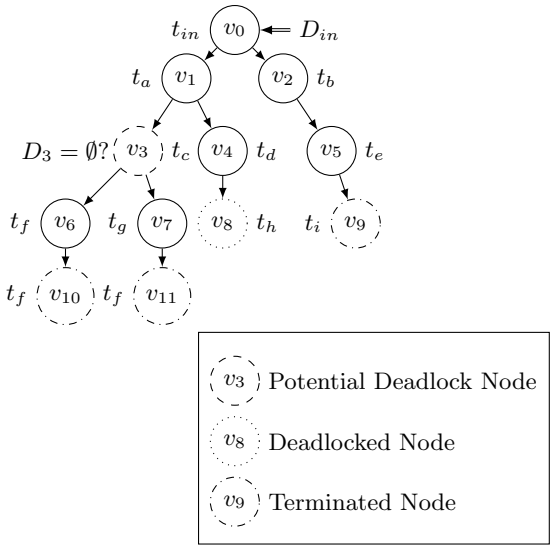

Fig. 10. Symbolic Execution Tree

these configurations (user assignments, task executions, refinements). Figure 10 represents a symbolic execution tree. Vertices of the tree are represented by circles. Deadlocked vertices are represented by dotted 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 stating 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 $X = \{x_1, \ldots x_k\}$ 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 $a_1.x_1 + a_2.x_2 + \ldots a_k.x_k \leq b$. 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 γ has a solution if and only if γ' has a solution. The algorithm works in three steps:

Step 1: Normalize all inequalities in γ, i.e. rewrite every inequality containing x_i of the form

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

into a new inequality of the form

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

or

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

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

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

γ^- contains all inequalities of the form

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

and $\gamma^\emptyset$ all other inequalities that do not refer to x_i.
Step 3: create a new system of inequalities that contains $\gamma^\emptyset$ and, for each pair of inequalities

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

and

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

a fresh inequality of the form

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

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.