Analyzing Workflows implied by Instance-Dependent Access Rules

Toon Calders[1] Stijn Dekeyser[2] Jan Hidders[1] Jan Paredaens[1]
[1] University of Antwerp (Belgium) [2] University of Southern Queensland (Australia)
toon.calders@ua.ac.be dekeyser@usq.edu.au jan.hidders@ua.ac.be jan.paredaens@ua.ac.be

ABSTRACT

Recently proposed form-based web information systems liberate the capture and reuse of data in organizations by substituting the development of technical implementations of electronic forms for the conceptual modelling of forms' tree-structured schemas and their data access rules. Significantly, these instance-dependent rules also imply a workflow process associated to a form, eliminating the need for a costly workflow design phase. Instead, the workflows thus created in an ad hoc manner by unsophisticated end-users can be automatically analyzed, and incorrect forms rejected.

This paper examines fundamental correctness properties of workflows that are implied by instance-dependent access rules. Specifically, we study the decidability of the form completability property and the *semi-soundness* of a form's workflow. These problems are affected by a choice of constraints on the path language used to express access rules and completion formulas, and on the depth of the form's schema tree. Hence, we study these problems by examining them in the context of several different fragments determined by such constraints.

1. INTRODUCTION

In previous work [4] we presented a preliminary description of a novel web information system based on forms to solve a practical problem that many organizations deal with. The main goal of such a system is to enable staff to easily and securely capture data using web forms, and reuse data that others (possibly connected via other peers) have captured, provided such reuse does not violate security or privacy rules. Users should only concentrate on developing the schema and access rules of their forms, and let the system handle storage, security and other related issues. As such, the proposed system already complements the capabilities of commercial software such as Microsoft's InfoPath [1] and the new XForms [2] standard being defined by the W3C to replace the original HTML-based web forms.

More significantly, from a theoretical point of view there are many different, traditionally database-oriented formal aspects associated to these form-based web information systems (FB-WIS), including data and schema integration [3] and workflow analysis [6, 10]. This paper studies the latter: access rules included in a form's definition imply a workflow process that—rather simply put—describes the order in which data can be entered in a form. Hence, in this type of workflow systems, the data-flow implies the control-flow.

In contrast to existing workflow systems and research, an FB-WIS does not require complex initial design and subsequent modification phases for workflows. This establishes the motivation for our paper: workflow processes implied by forms that were created in an ad hoc manner by unsophisticated users need to be analyzed automatically such that forms with an incorrect workflow will be rejected by the FB-WIS and users can be told how they should modify their form's definition.

Importantly, the problems we study in this paper do not only occur in the form-based WIS setting. Indeed, even relational database management systems suitably enhanced with instance-dependent access rules imply workflows. While the problem area is therefore a general one, we will use the FB-WIS setting as the practical application in which to study it, since that setting is particularly well-suited to solve e-government issues.

An example of a complex form used in this context is the electronic version of a tax declaration: various parts of the e-form may only be completed by certain persons and then only depending on information that has already been entered. The associated workflow process can be complex, especially when the form allows interaction between citizens and various levels of the administrative bureaucracy that processes the data captured through the form.

Model and Fragments. The general model we use for the FB-WIS employs nested relations as the schema for forms, and a subset of XPath's abbreviated syntax for access rules and completion formulas. Restrictions on either (or both) the schema and the formula languages constitute various fragments that we examine in depth. For instance, we will at times restrict the depth of a schema (as in the *depth-k* fragment) or prohibit the use of negation in either access or completion formulas (as in the *positive* fragments).

Note that the original model we proposed for peer-to-peer FB-WIS [4] added ISA constraints to create links between multiple forms' schemas to enable reuse of data across peers. We have omitted this feature in this paper since we will concentrate on a single workflow implied by a single form.

Problems. The restrictions employed in the various fragments facilitate the study of some facets of the general problem which examines the properties of workflows implied by instance-dependent access rules. We consider the decidability of a form's correctness, one aspect of which we call the *completability problem* because it corresponds to determining whether a form can be completed (according to some formula set by the form's creator) given its access rules. A related question, the *semi-soundness problem*[1], is to decide whether each reachable instance is completable.

Running Example. Consider that a web form exists that enables staff to apply for recreational leave and managers to decide on all applications. The Leave Application form has a schema and data access rules that allow a person to create new instances and update details such as start and end dates. When the user submits the application (e.g. by checking the appropriate box), those details cannot be changed anymore, and the form (or rather that instance of the form) becomes accessible by the manager. The latter can approve or reject the leave application, possibly giving a motivation for doing so. The form is completed by checking a final check box, after which nothing can be changed. Hence, the form completion formula simply checks the value of this check box.

Structure. Section 2 discusses relevant research, while the remainder of this paper is divided in two parts. Section 3 formally defines the FB-WIS model, the fragments we study and the completability and semi-soundness problems. Sections 4 and 5 examine the complexity of deciding these properties for forms with unrestricted access rules and form completion formulas, and with only positive formulas, respectively. We wrap up with a short conclusion.

2. RELATED WORK

2.1 Practical Work

The World Wide Web Consortium's *XForms* [2] recommendation is a specification for a declarative language used to specify web forms—which may include data constraints and conditional formatting logic—based on XML Schema and XPath. Compared to traditional HTML forms, XForms provide serialization of captured data to XML, separate presentation from content, minimize round-trips to the server, offer device independence, and reduce the need for scripting. The main differences with FB-WIS are that an FB-WIS is an active server-side system that (1) frees the form creator from having to explicitly specify access to a database back-end; (2) includes data access rules in the form definitions enabling a more fine-grained security specification; and (3) implies and manages a workflow process automatically. As XForms complements the functionality of an FB-WIS, the latter will normally make use of XForms for client-side constraint checking and serialization of data.

Microsoft's *InfoPath* [1] software (MSIP) is an alternative to XForms but simplifies "conditional formatting" and makes its use explicit. While the focus is on presentation, conditional formatting uses a powerful path expression language to decide whether or not a specific field in a form should be made visible to the user. This is usually determined on the basis of data already entered in the form. Hence, this technique is similar to the instance-dependent data access rules we propose for FB-WIS. The practical differences between the systems are: (1) our data access rules are simple path expressions making the study of workflow correctness feasible; (2) MSIP's rules determine presentation issues in the client, not data access in the storage back-end; and (3) form creators in MSIP, as in XForms, are tasked with explicitly creating access to data sources.

Of interest to this paper, both MSIP and XForms are popular tools that allow unsophisticated users to imply a workflow by using conditional formatting in their forms, but do not analyze the correctness of these workflows. Hence, from a theoretical point of view, both systems give rise to the same set of workflow-related problems as those implied by an FB-WIS. Some of these problems are studied in this paper.

2.2 Theoretical Work

The idea to let access rules implicitly define a workflow is partly inspired by Product-Based Workflow Design as introduced by Hajo Reijers et al. [7] where a method is proposed to design workflows based upon the final product, similar to a form, and dependencies between the different pieces of information in the form. There has also already been a significant amount of research on specifying access control policies for XML documents in terms of XPath expressions (see Fundulaki and Marx [5] for an overview) but to the best of our knowledge none of them attempt to analyse the workflow that is implied by these access control policies.

3. PRELIMINARIES

3.1 Schemas

Informally a *schema* defines the underlying data structure of a form. It is basically a schema of a nested relation as defined in the nested relational model [8]. For example, in the leave application, a possible schema could look like the model given in Figure 1.

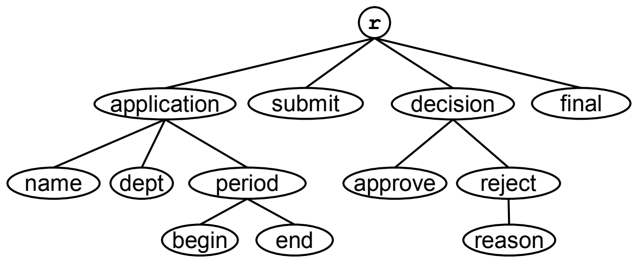


Figure 1: The leave application schema.

We assume the existence of a set $\mathcal{L}$ of node labels, and a special label $\mathtt{r} \in \mathcal{L}$ that will be used to label the roots of trees. In the following of this paper we will abbreviate the node labels in Figure 1 to their first letter, i.e., application to a, name to n, et cetera.

[1]This property is called semi-soundness because it is a weaker version of the usual notion of soundness for workflow nets [9] which also requires that each event occurs in at least one possible run of the workflow.

DEFINITION 3.1 (SCHEMA AND INSTANCE). *A* rooted node-labelled tree *is a four-tuple $M = (V, E, r, \lambda)$ where the graph (V, E) is a tree with root $r \in V$, and $\lambda : V \to \mathcal{L}$ is a labelling function on the nodes.*

A schema *is a rooted node-labelled tree in which no two siblings have the same label, and the root has label* `r`*.*

A rooted node-labelled tree $I = (V', E', r', \lambda')$ *is called an* instance *of the schema $M = (V, E, r, \lambda)$ if there exists a homomorphism from I to M. That is, there exists a function $h : V' \to V$ such that (1) for all $v_1, v_2 \in V$ it holds that if $(v_1, v_2) \in E'$, then $(h(v_1), h(v_2)) \in E$, (2) $h(r') = r$, and (3) for all $v' \in V'$, $\lambda'(v') = \lambda(h(v'))$.*

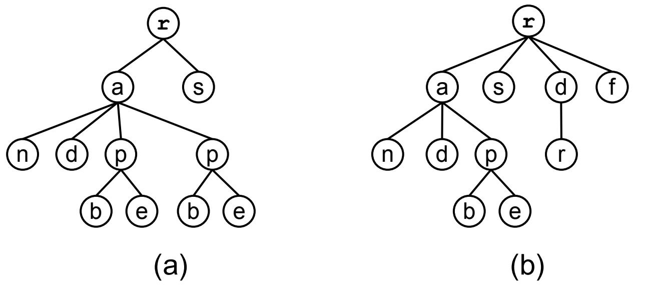


Figure 2: Instances of the leave application schema.

EXAMPLE 3.2. *In Figure 2 two examples of instances of the schema in Figure 1 are given. The instance (a) shows a submitted application for two periods, and (b) an application for a single period that was rejected. Unless explicitly indicated the fields in a form can contain zero or more elements. It will be discussed later how fields can be made single-valued by choosing appropriate access rules. Note that in practice there will be values associated with the fields such as the actual name, department and the dates, but this is beyond the scope of this paper.*

PROPOSITION 3.3. *If a form I is an instance of a schema M, then the homomorphism from I to M is unique.*

Hence, every node v and edge e in an instance I of schema M are always associated with a single node and edge in M. We denote this unique node and unique edge as $\hat{v}$ and $\hat{e}$, respectively.

3.2 Formulas

DEFINITION 3.4 (FORMULA). *Formulas are defined by the following abstract syntax:*

$$\begin{aligned} F &::= P \mid \neg F \mid (F \wedge F) \mid (F \vee F) \\ P &::= \,..\, \mid \mathcal{L} \mid (P/P) \mid P[F]. \end{aligned}$$

We will use formulas to specify *access* and *update rules* and *form completion formulas* for forms.

The abstract syntax closely resembles XPath's abbreviated syntax: the `descendent-or-self` (//) and `self` (.) axes are omitted, the former because the schema's depth—and hence that of its instances—is fixed when it is created, and the latter because it does not add expressive power in this context. In [4] we also allowed any conditions in formulas to include comparisons to the system variables 'user-id' and 'datetime'. In this paper we assume only one user; the data access formulas are evaluated for this user and do not include temporal conditions.

DEFINITION 3.5 (SEMANTICS OF FORMULAS). *The expression $n \models_T \phi$ indicates that the formula ϕ is true for the rooted node-labelled tree $T = (V, E, r, \lambda)$ and a node n in T. The expression $n \xrightarrow{p}_T n'$ indicates that the path expression p is true for the rooted node-labelled tree T, a begin node n, and an end node n' in T. We will omit the subscript T in both cases if it is clear from the context.*

The following rules define both concepts:

$$\begin{aligned}
n \models p &\Leftrightarrow \text{there exists a } n' \in V \\
& \text{such that } n \xrightarrow{p} n' \\
n \models \neg\phi &\Leftrightarrow n \models \phi \text{ does not hold} \\
n \models \phi \wedge \psi &\Leftrightarrow (n \models \phi) \text{ and } (n \models \psi) \\
n \models \phi \vee \psi &\Leftrightarrow (n \models \phi) \text{ or } (n \models \psi) \\
n \xrightarrow{..} n' &\Leftrightarrow (n', n) \in E \\
n \xrightarrow{l} n' &\Leftrightarrow (n, n') \in E \text{ and } \lambda(n') = l \\
n \xrightarrow{p/q} n' &\Leftrightarrow \text{there exists a } n'' \in V \\
& \text{such that } n \xrightarrow{p} n'' \text{ and } n'' \xrightarrow{q} n' \\
n \xrightarrow{p[\phi]} n' &\Leftrightarrow n \xrightarrow{p} n' \text{ and } n' \models \phi
\end{aligned}$$

EXAMPLE 3.6. *We give some examples of formulas for instances of the schema in Figure 1. They are assumed to be evaluated for the root* `r`*:*
$\neg$a/p[$\neg$b $\vee$ $\neg$e] *: All periods have begin and end dates.*
$\neg$f $\vee$ d[a $\vee$ r] *: The application cannot be final unless it was rejected or approved.*
d[$\neg$(a $\wedge$ r)] *: The application cannot be both rejected and approved.*

3.3 Canonical Instance

We define a notion of formula equivalence that can be informally described as bisimulation under the assumption that all edges are bidirectional.

DEFINITION 3.7 (FORMULA EQUIVALENCE). *A* formula equivalence *between two form instances $I = (V, E, r, \lambda)$ and $I' = (V', E', r', \lambda')$ is a relation $R \subseteq V \times V'$, such that:*

- *$(r, r') \in R$;*
- *for all $(v, v') \in R$: $\lambda(v) = \lambda(v')$;*
- *for all $(v, w) \in E$: if $(w, w') \in R$, then there exists a $v' \in V'$ such that $(v, v') \in R$, and $(v', w') \in E'$;*
- *for all $(v', w') \in E'$: if $(w, w') \in R$, then there exists a $v \in V$ such that $(v, v') \in R$, and $(v, w) \in E$;*
- *for all $(v, w) \in E$: if $(v, v') \in R$, then there exists a $w' \in V'$ such that $(w, w') \in R$, and $(v', w') \in E'$; and,*
- *for all $(v', w') \in E'$: if $(v, v') \in R$, then there exists a $w \in V$ such that $(w, w') \in R$, and $(v, w) \in E$.*

If there exists a formula equivalence between I_1 and I_2, we call them formula equivalent*, and this is denoted $I_1 \sim I_2$.*

Two nodes v, v' of a form instance I are called formula equivalent nodes *if there exists a formula equivalence R between I and itself, such that $(v, v') \in R$.*

It is well-known that bisimilarity between instances, and between the nodes of an instance are equivalence relations between respectively the instances, and the nodes of an instance, so this also holds for formula equivalence. Furthermore, if two instances are formula equivalent, all formulas will evaluate to the same truth value on them. For every class of formula equivalent instances, we can isolate a single *canonical* (up to isomorphism) instance as is indicated in the following definition and lemma.

DEFINITION 3.8 (CANONICAL INSTANCE). *Let I be an instance. Let $[v]$ denote the equivalence class of a node v. The* canonical instance *$can(I)$ is the following form instance: $([V],[E],[r],[\lambda])$, with nodes $[V] = \{[v] \mid v \in V\}$, edges $[E] = \{([v],[w]) \mid (v,w) \in E\}$, and $[\lambda]([v]) = \lambda(v)$.*

Notice that $[\lambda]$ is well-defined, because every pair of formula equivalent nodes must have the same label. Moreover, the result is a tree because if two nodes in I are formula equivalent then they either are both the root r or both have parents which are formula equivalent.

LEMMA 3.9. *Let I and J be two form instances. If R is a formula equivalence between I and J and $(v,w) \in R$ then for all formulas φ, $v \models_I \varphi$ iff $w \models_J \varphi$. Furthermore, $I \sim can(I)$, and $can(I)$ is isomorphic to $can(J)$.*

EXAMPLE 3.10. *In Figure 3 we see an example of an instance (a) and the corresponding canonical instance (b).*

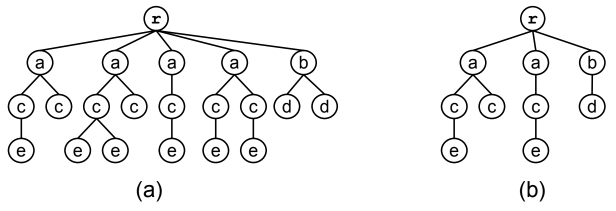


Figure 3: An instance and corresponding canonical instance.

3.4 Access and Update Rules

We postulate the set of *access rights* $\mathcal{R} = \{\mathsf{add}, \mathsf{del}\}$ where add and del are the right to create and delete respectively.

DEFINITION 3.11 (GUARDED FORM). *A guarded form is a tuple (M, A, I_0, φ) where $M = (V, E, r, \lambda)$ is a schema, $A : \mathcal{R} \times E \to F$ the access-rule function that maps each access right and edge in M to a formula, I_0 the initial instance that is an instance of M, and φ a* completion formula *that defines when the form is complete by being true for the root node.*

The only updates on instance trees that are considered are the additions and deletions of edges that add and remove leaf nodes, respectively. The addition of an edge $e = (n, n')$ is allowed iff the formula $A(\mathsf{add}, \hat{e})$ is true for n in the current instance. Similarly, the deletion of an edge $e = (n, n')$ is allowed iff the formula $A(\mathsf{del}, \hat{e})$ is true for n in the current instance. Given a guarded form (M, A, I_0, φ) we call a sequence $I_0, \ldots, I_n$ of instances of M a *run* of this guarded form if for every $1 \leq i \leq n$ it holds that I_i can be obtained from I_{i-1} by a single edge addition or deletion which is allowed by A for I_{i-1}, and a *complete run* if I_n satisfies φ.

Note that we do not consider *read* or *update* rights here because to the extent that these are relevant for workflow analysis these can be simulated in the presented formalism. For example, it might be that the user can only see (and therefore update) a node if he or she has read-rights on this node and all its ancestors, but this can be simulated by adding the formulas of the read rights to the formulas that determine the create and delete-rights.

EXAMPLE 3.12. *To illustrate the concept of guarded form take the schema M in Figure 1, the initial instance I_0 that consists only of the root r (we start with an empty form), a completion formula $\varphi = \mathsf{f}$ (the final field has been marked) and the access-rule function A as follows (schema edges are identified by the paths to their end nodes):*

$$
\begin{array}{llll}
A(\mathsf{add},\mathsf{a}) & = \neg\mathsf{a} & A(\mathsf{del},\mathsf{a}) & = \neg\mathsf{a} \\
A(\mathsf{add},\mathsf{a/n}) & = \neg\mathsf{../s} \wedge \neg\mathsf{n} & A(\mathsf{del},\mathsf{a/n}) & = \neg\mathsf{../s} \\
A(\mathsf{add},\mathsf{a/d}) & = \neg\mathsf{../s} \wedge \neg\mathsf{d} & A(\mathsf{del},\mathsf{a/d}) & = \neg\mathsf{../s} \\
A(\mathsf{add},\mathsf{a/p}) & = \neg\mathsf{../s} & A(\mathsf{del},\mathsf{a/p}) & = \neg\mathsf{../s} \\
A(\mathsf{add},\mathsf{a/p/b}) & = \neg\mathsf{../../s} \wedge \neg\mathsf{b} & A(\mathsf{del},\mathsf{a/p/b}) & = \neg\mathsf{../../s} \\
A(\mathsf{add},\mathsf{a/p/e}) & = \neg\mathsf{../../s} \wedge \neg\mathsf{e} & A(\mathsf{del},\mathsf{a/p/e}) & = \neg\mathsf{../../s} \\
A(\mathsf{add},\mathsf{s}) & \multicolumn{3}{l}{= \neg\mathsf{s} \wedge \mathsf{a}[\mathsf{n} \wedge \mathsf{d} \wedge \mathsf{p}] \wedge \neg\mathsf{a/p}[\neg\mathsf{b} \vee \neg\mathsf{e}]} \\
 & & A(\mathsf{del},\mathsf{s}) & = \neg\mathsf{s} \\
A(\mathsf{add},\mathsf{d}) & = \mathsf{s} \wedge \neg\mathsf{d} & A(\mathsf{del},\mathsf{d}) & = \neg\mathsf{f} \\
A(\mathsf{add},\mathsf{d/a}) & = \neg(\mathsf{a} \vee \mathsf{r}) & A(\mathsf{del},\mathsf{d/a}) & = \neg\mathsf{../f} \\
A(\mathsf{add},\mathsf{d/r}) & = \neg(\mathsf{a} \vee \mathsf{r}) & A(\mathsf{del},\mathsf{d/r}) & = \neg\mathsf{../f} \\
A(\mathsf{add},\mathsf{d/r/r}) & = \neg\mathsf{r} & A(\mathsf{del},\mathsf{d/r/r}) & = \neg\mathsf{../../f} \\
A(\mathsf{add},\mathsf{f}) & = \mathsf{d}[\mathsf{a} \vee \mathsf{r}] \wedge \neg\mathsf{f} & A(\mathsf{del},\mathsf{f}) & = \neg\mathsf{f}
\end{array}
$$

In Example 3.12 $A(\mathsf{add}, \mathsf{a})$ requires that there is not already an a field present and so there cannot be two applications in an instance. The rule $A(\mathsf{del}, \mathsf{a})$ requires that there is no a field so we can never delete an application field once it has been added. For the edge $\mathsf{a/n}$ there are similar rules except that it is also required that there is no s field yet, i.e, the application was not yet submitted. Note that since the formula is evaluated for the a node and not the root $\mathtt{r}$ it reads $\neg\mathsf{../s}$ and not just $\neg\mathsf{s}$. For adding an $\mathsf{a/p}$ edge it is not required $\neg\mathsf{p}$ and so there can be more than one period in an application. This is reflected in $A(\mathsf{add}, \mathsf{s})$ where it is required that before submission there not only must be a name, $\mathsf{department}$ and at least one period, but it must also hold that there is no period without begin or end date.

3.5 Completability and Semi-soundness

To illustrate the problems that we will study consider the guarded from in Example 3.12 except that $\varphi = \mathsf{f} \wedge \neg\mathsf{s}$. It can be observed that if we start from the initial instance there is no full run, i.e., we can never reach an instance that satisfies φ via a sequence of allowed updates. Clearly such a guarded form cannot be considered correct, so we define the following problem.

DEFINITION 3.13 (COMPLETABILITY PROBLEM). *For a guarded form $G = (M, A, I_0, \varphi)$ the* form completability problem *is to decide if there exists at least one complete run of G.*

Note that completability is not only interesting as a correctness requirement but also important for deciding invariants. For example, by checking completability for $\varphi = \mathsf{d}[\mathsf{a} \wedge \mathsf{r}]$ we can check if at any stage there can be a $\mathsf{decision}$ field that contains both accept and reject.

If a guarded form is completable it may still have completion problems. Consider again Example 3.12 but with $\varphi = \mathsf{f} \wedge \mathsf{d}[\mathsf{a} \vee \mathsf{r}]$, $A(\mathsf{add}, \mathsf{f}) = \mathsf{d} \wedge \neg\mathsf{f}$, $A(\mathsf{add}, \mathsf{d/a}) = \neg(\mathsf{a} \vee \mathsf{r}) \wedge \neg\mathsf{../f}$ and $A(\mathsf{add}, \mathsf{d/r}) = \neg(\mathsf{a} \vee \mathsf{r}) \wedge \neg\mathsf{../f}$. In this case the guarded form is still completable but at the same time it is possible to reach an instance where there is a final field but no approval or reject field. From that instance the form cannot be completed because the final field prevents the addition of the required approval/reject field. Obviously, if form completability is a decidable problem, a form manager might disallow any updates that lead to such an instance from which completion is not possible, but in a well-designed guarded form the access rules should prevent this. This leads to the following problem.

Definition 3.14 (Semi-soundness Problem). *Given a guarded form $G = (M, A, I_0, \varphi)$ the* form semi-soundness problem *is to decide if for every run $I_0, \ldots, I_n$ of G it holds that (M, A, I_n, φ) is completable.*

In the following sections, we study the complexity of deciding the completability and the semi-soundness of guarded forms. For the general case, we show that these problems are both undecidable. Therefore, we also study more restricted fragments of guarded forms, for which these properties are decidable.

The restrictions are threefold: we study the cases in which (a) the access rules do not contain negation (A^+), (b) the completion formula does not contain negation (φ^+), and (c) the depth is limited to either 1, or a fixed constant k. We also study all possible combinations of these restrictions. We will denote these classes of fragments by $F(A, \varphi, d)$, with A either A^+ or A^-, denoting respectively only positive access rules or any access rules; φ is either φ^+ or φ^-, denoting respectively a positive completion formula, or an unrestricted completion formula; and d being 1, k, or ∞, denoting that the depth of the schemas is respectively 1 (e.g., under the root there is at most 1 level of nodes), a fixed constant k, or unrestricted. For example, $F(A^+, \varphi^-, 1)$ denotes the class of guarded forms where all access rules are positive formulas, the completion formula can contain negation, and the depth of the schema is at most 1. The class without restrictions is denoted $F(A^-, \varphi^-, \infty)$.

4. UNRESTRICTED ACCESS RULES

4.1 Unrestricted completion formulas

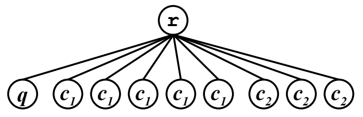
We will show that within the context of the form completability problem we can simulate a two-counter machine. It is well-known that two-counter machines are Turing complete, and that their halting problem is undecidable. From the simulation it is straightforward, given a two-counter machine, to create an instance of the form completability problem that is completable if and only if the two-counter machine will eventually halt when the empty string is given as input. This property of two-counter machines is undecidable.

A two-counter machine is a deterministic state machine with two counters. The state transitions depend only on the input symbol read, the current state, and on the counters being zero. Furthermore, during a transition the counters can be incremented or decremented by 1, independently, and the input cursor can either stay in place, or advance one position. Because we only consider the problem of deciding whether or not the two-counter machine halts on the empty string, we do not have to take care of the input-cursor. Thus, a two-counter machine without input can be modelled as a three-tuple (Q, F, δ), with Q a finite set of states, $F \subseteq Q$ the set of accepting states, and δ the transition function that maps $Q \times \{0, +\} \times \{0, +\}$ to $Q \times \{-, 0, +\} \times \{-, 0, +\}$. For example, $\delta(q, +, +) = (p, -, 0)$ denotes that if the two-counter machine is in state q, and both counters are nonzero, the state will change to p, the first counter will be decremented, and the second remains unchanged. A configuration of a two-counter machine without input will be denoted as a three-tuple (q, n, m) with q the state, and n and m positive integers that indicate the respective contents of the two counters.

Theorem 4.1. *For guarded forms in $F(A^-, \varphi^-, \infty)$ the completability and semi-soundness problems are undecidable, even if only forms of depth 2 are considered.*

Proof. The proof proceeds by showing that the configuration of an inputless two-counter machine can be represented, and how we can move from one configuration to the next by discussing the procedures for decrementing and incrementing the counters. Therefore, we can simulate the halting problem of an inputless two-counter machine with the completability of a guarded form. Because this property is undecidable, the completability problem is undecidable as well. Furthermore, the two-counter machines we consider are deterministic. As a consequence, in every reachable instance of the guarded form we construct, only one other instance (up to duplications of nodes) is reachable. Thus, in this case, the completability problem and the semi-soundness problem are equivalent. Notice incidentally that the depth of the guarded form constructed below is always 2, independent of the simulated two-counter machine.

Configuration: Let (q, n, m) be a configuration of a two-counter machine. This configuration will in the simulation be represented by the following instance $Conf(q, n, m)$: below the root there is a node labelled q, n nodes labelled c_1 and m nodes labelled c_2. For example, the following instance denotes the configuration $(q, 5, 3)$.



Transitions: The most complex part of the reduction is to give an access-right function, such that we can guarantee that when moving from one configuration tree to another, we use the transition function δ. Once we can ensure this guarantee, the stopping condition in the form will simply be the disjunction of all accepting states.

For every possible transition $\delta(q, s_1, s_2) = (p, a_1, a_2)$, a new node $init(q, s_1, s_2)$ is introduced in the schema of the guarded form. The presence of this node in an instance will denote that the transition $init(q, s_1, s_2)$ is in progress. This is necessary in order to avoid that the execution of one transition already starts before the previous transition is completely finished. To achieve this, the addition condition for $init(q, s_1, s_2)$ will be:

$$q \wedge \sigma_1 \wedge \sigma_2 \wedge \bigwedge_{\substack{p \in Q, t_1 \in \{0,+\}, t_2 \in \{0,+\} \\ (p, t_1, t_2) \neq (q, s_1, s_2)}} \neg init(p, t_1, t_2) \ ,$$

where σ_i is c_i (resp. $\neg c_i$) if s_i is $+$ (resp. 0). That is, the transition $init(q, s_1, s_2)$ can only be initiated if we are in state q, there is a c_i iff s_i is $+$ (indicating that counter i is strictly positive), and none of the other transitions is still in progress.

Increments: For the increment of counter 1, we need to add exactly one c_1 below the root. We have to be careful in this case, because we want to avoid that more than one c_1 can be added. Therefore, we need to be able to make a difference between the situation before the addition, and after. This can be done as follows: first we mark all c_1-nodes, by adding a node labelled d below them and adding a special node m once this is done for all of them. Then, one c_1 is added. Notice that we can clearly make a distinction between the situation before the addition, and the situation afterwards; before the addition all c_1 nodes are marked; hence $\neg(c_1[\neg d])$ is true, and after the addition, there is one unmarked node; hence $c_1[\neg d]$) is true). The action is then completed by removing all marks, and moving to the next state.

What follows illustrates some of the corresponding access rules if δ consists only of the transition $\delta(q_0, 0, +) = (q_1, +, 0)$:

$$
\begin{aligned}
A(\mathsf{add}, init(q_0,0,+)) &= q_0 \wedge \neg c_1 \wedge c_2 \wedge \neg init(q_0,0,0) \wedge \\
&\quad \neg init(q_0,+,0) \wedge \neg init(q_0,+,+) \wedge \\
&\quad \neg init(q_1,0,0) \wedge \neg init(q_1,0,+) \wedge \\
&\quad \neg init(q_1,+,0) \wedge \neg init(q_1,+,+) \\
A(\mathsf{add}, c_1/d) &= ../init(q_0,+,0) \\
A(\mathsf{add}, m) &= init(q_0,+,0) \wedge \neg c_1[\neg d] \\
A(\mathsf{add}, c_1) &= init(q_0,+,0) \wedge m \wedge \neg c_1[\neg d] \\
A(\mathsf{del}, c_1/d) &= ../init(q_0,+,0) \wedge ../m \\
A(\mathsf{del}, m) &= init(q_0,+,0) \wedge \neg c_1[d] \\
A(\mathsf{add}, q_1) &= init(q_0,+,0) \\
A(\mathsf{del}, q_0) &= init(q_0,+,0) \\
A(\mathsf{del}, init(q_0,0,+)) &= init(q_0,+,0) \wedge \neg c_1[d] \wedge \neg m \wedge \\
&\quad q_1 \wedge \neg q_0
\end{aligned}
$$

Decrements: The decrement is the most difficult operation to translate. A first attempt could be to mark exactly one c_1-node, and subsequently remove the marked node. This is, however, impossible, as we can only do operations on leaves of the instance. A second attempt could be to mark all nodes, except for one. But, this is not at all straightforward, because at the moment that we need to decide whether or not to add an additional mark, we cannot distinguish between the situation where one node is unmarked, or where two nodes are unmarked. Therefore, the following, rather cumbersome, procedure is followed. First we mark exactly one c_1-node by adding a node labelled d below it. This is the node that will be deleted later on. Then, we mark every other c_1-node with d'. Subsequently, we unmark the node with mark d and then remove the sole unmarked c_1-node (this is the node that was marked with d before.) Finally, we remove all marks d'. $\square$

It might be suspected that the presence of deletions makes the problem hard but the following corollary shows otherwise.

COROLLARY 4.2. *For the class $F(A^-, \varphi^-, \infty)$ the completability problem and semi-soundness problem are undecidable, even if only additions and forms of depth 3 are considered.*

PROOF. The proof is similar to that of Theorem 4.1 except that (1) every deletion of an edge is replaced with the addition of an edge under that edge that ends in a node with a special label, say $\mathsf{deleted}$, and (2) in all formulas we replace path expressions of the from l with $l[\neg\mathsf{deleted}]$. $\square$

LEMMA 4.3. *Let G be a guarded form in $F(A^-, \varphi^-, 1)$ and let C be a canonical instance of the schema of G. There exists an instance J, reachable from I, with $can(J) = C$, if and only if C is reachable from $can(I)$.*

Furthermore, in a guarded form of depth 1, instance I can be completed if and only if $can(I)$ can be completed.

PROOF. Every addition that is allowed in I, is also allowed in $can(I)$. As such, every operation on I, resulting in an instance J with a different canonical form, can be imitated on $can(I)$ to get $can(J)$, and vice versa. $\square$

LEMMA 4.4. *Let φ be an arbitrary formula, and let T be a rooted tree that satisfies φ, then there exists an embedded subtree T' of T that also satisfies φ and has a branching factor that is linear in the length of φ.*

PROOF. We start with the observation that the following equivalencies hold, where $\psi_1 \equiv \psi_2$ means that for every instance I and node n in I it holds that $n \models_I \psi_1 \Leftrightarrow n \models_I \psi_2$:

$$
\begin{aligned}
(p_1/p_2)[\psi] &\equiv p_1[p_2[\psi]] \\
(p_1[\psi_1])[\psi_2] &\equiv p_1[\psi_1 \wedge \psi_2] \\
(p_1/p_2)/p_3 &\equiv p_1/(p_2/p_3) \\
(p_1[\psi])/p_2 &\equiv p_1[\psi \wedge p_2] \\
l/p &\equiv l[p] \\
../p &\equiv ..[p]
\end{aligned}
$$

By applying these rules from left to right to φ and its subformulas we can obtain a formula φ' that is linear in the size of φ and belongs to the following syntax:

$$
\begin{aligned}
F' &::= P' \mid \neg F' \mid F' \wedge F' \mid F' \vee F' \\
P' &::= \mathcal{L} \mid .. \mid \mathcal{L}[F'] \mid ..[F']
\end{aligned}
$$

Given a formula ψ of the from F' we define a *selection* of ψ as a set S of sub-formulas and negated sub-formulas of ψ such that (1) $S = \{p\}$ if $\psi = p$ with p in P', (1) $S = \{\neg p\}$ if $\psi = \neg p$ with p in P', (3) $S = S'$ if $\psi = \neg(\psi_1 \wedge \psi_2)$ and S' is a selection of $\neg\psi_1 \vee \neg\psi_2$, (4) $S = S'$ if $\psi = \neg(\psi_1 \vee \psi_2)$ and S' is a selection of $\neg\psi_1 \wedge \neg\psi_2$, (5) $S = S_1 \cup S_2$ where $\psi = \psi_1 \wedge \psi_2$ and S_1 and S_2 are selections of ψ_1 and ψ_2, respectively and (6) $S = S_1$ or $S = S_2$ where $\psi = \psi_1 \vee \psi_2$ and S_1 and S_2 are selections of ψ_1 and ψ_2, respectively. Observe that a node n in an instance I satisfies a formula ψ iff there is a selection of ψ such that all formulas in this selection are satisfied by n in I. If this holds then we say that the selection is satisified by node n in I.

From φ' and T we construct T' that satisfies φ' as follows. We start by letting T' be the node n for which ϕ holds in T and associate with n a set $\Phi(n)$ which is a selection of φ' that was satisfied by n in T. Then we proceed by applying the following rules:

- For each formula of the form $l[\psi']$ in $\Phi(n')$ we add to T' a child n'' of n' with label l that made this formula true for n' in T, unless such a child is already present in T', and add to $\Phi(n'')$ a selection of ψ' that is satisfied by n'' in T, and for each formula of the form l in $\Phi(n')$ we add to T' a child n'' with label l.

- For each formula of the form $\neg l[\psi']$ in $\Phi(n')$ and in T' each child n'' of n' with label l we add a selection of $\neg\psi'$ that was was satisfied for n'' in T to $\Phi(n'')$, and if there is a formula of the form $\neg l$ in $\Phi(n')$ and in T' a child n'' of n' with label l then we abort the construction.
- For each formula of the form $..[\psi']$ in $\Phi(n')$ we add to T' the parent n'' of n', if it was not already in T', and add to $\Phi(n'')$ a selection of ψ' that is satisfied by n'' in T, and for each formula of the form .. in $\Phi(n')$ we add to T' a parent n'' of n', if it was not already in T'.
- For each formula of the form $\neg..[\psi']$ in $\Phi(n')$ and parent n'' of n' we add n'' to T', if it was not already in T', and add to $\Phi(n'')$ a selection of $\neg\psi'$ that is satisfied by n'' in T, and if there is a formula of the form $\neg..$ in $\Phi(n')$ and a parent n'' of n' then we abort the construction.

It is easy to see that if φ' was satisfied by n in T then the construction of T' will not abort and φ' will be satisfied by n in T'. Moreover, because of the way child nodes are added the fan-out of each node is restricted by the number of sub-formulas of φ' and hence linear in the size of φ. $\Box$

COROLLARY 4.5. *Testing satisfiability of a formula φ is* **NP***-complete if the depth of the instances is limited by a constant k, and is* **PSPACE***-complete if the depth is unlimited.*

PROOF. The **NP** upper bound follows from Lemma 4.4 since it shows that there is a succinct certificate of size $O(n^k)$ for the satisfiability of φ if n is the size of φ

The **NP**-hardness proof is a straightforward reduction from SAT to satisfiability; e.g., the satisfiability of the propositional formula $(x_1 \vee x_2) \wedge \neg x_3$ corresponds to the satisfiability of the formula $(a \vee b) \wedge \neg c$.

The **PSPACE** upper bound can be shown as follows. We transform φ to φ' and use the same generation rules as in the proof of Lemma 4.4. Then we follow the same procedure except that (1) we do not generate the whole tree but only walk through it depth-first and (2) with each node n'' that is added we also guess immediately the set $\Phi(n'')$ and check if it consistent with that of n'. Since the size of $\Phi(n'')$ is bounded by the size of φ' it follows that for each node in the tree we have to do a polynomial amount of work. Moreover, since the depth of the tree cannot become larger then the size of φ' it follows that the stack that is required for the depth-first tree-walk is polynomial in the size of φ'.

The **PSPACE**-hardness can be shown with a reduction from QSAT to satisfiability. E.g., the satisfiability of the quantified Boolean formula $\exists x \forall y \exists z : (x \vee y \wedge \neg z)$ corresponds to the satisfiability of the path expression

$$(\neg a_x/a_y/a_z[\neg(../../x) \vee (../y) \wedge \neg z]) \tag{4.1}$$
$$\wedge(a_x/x \leftrightarrow (\neg a_x[\neg x])) \tag{4.2}$$
$$\wedge(\neg(a_x[\neg a_y/y])) \wedge (\neg(a_x[\neg a_y[\neg y]])) \tag{4.3}$$
$$\wedge(a_x/a_y[a_z/z \leftrightarrow (\neg a_z[\neg z])]) \tag{4.4}$$

In this path expression, assignments for x are encoded with an a_x-node. The assignment making x true is encoded as a node a_x with a subnode x. Similarly for a_y and a_z. The assignments are nested to represent the quantifier order. Furthermore, line (4.2) makes sure that there is only one truth assignment for x (possibly represented by multiple identical copies). Line (4.3) ensures that within every a_x-node, there is an assignment a_y making y true, and one making y false. Line (4.4) ensures again that a within every a_y-node a unique choice for z is made. Finally, line (4.1) expresses that on every path, the given formula must be true. $\Box$

THEOREM 4.6. *Deciding completability for $F(A^-, \varphi^-, 1)$ is* **PSPACE***-complete.*

PROOF. The upper bound follows from Lemma 4.3 since it implies that we can decide completability in **NPSPACE** by guessing a sequence of canonical instances of the schema such that (1) the first is $can(I_0)$ where I_0 is the initial instance, (2) each next instance can be obtained from the preceding one by applying one or more times a certain addition or deletion that is allowed and (3) the last instance satisfies the completion formula.

We show **PSPACE**-hardness by reducing the **PSPACE**-complete problem *reachable deadlock* to the completability problem.

The reachable deadlock problem is the following. The input consists of a list of graphs $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ with disjoint sets of vertices, a sequence of vertices $v_1, \ldots, v_k$, with $v_i \in V_i$, $i = 1 \ldots k$, and a set T of pairs of edges (e_i, e_j) with e_i and e_j in different graphs. A configuration is any set $a_1, \ldots, a_k$ with $a_1 \in V_1, \ldots, a_k \in V_k$. There is a transition of configuration $a_1, \ldots, a_k$ to configuration $b_1, \ldots, b_k$, if there exist two indices $1 \leq i < j \leq k$ such that for every $\ell = 1 \ldots k$, $\ell \neq i, j$, $a_\ell = b_\ell$, and $((a_i, a_j), (b_i, b_j))$ is in T. We then call $b_1, \ldots, b_k$ a successor of $a_1, \ldots, a_k$. The reachable deadlock problem now is: does there exist a configuration reachable from $v_1, \ldots, v_k$ that does not have a successor?

We reduce the reachable deadlock problem to the form completability problem for depth one. For every vertex v in $V_1 \cup \ldots \cup V_k$, and for every transition $t \in T$, there is a node $n(v)$ respectively $n(t)$ in the schema. A configuration $a_1, \ldots, a_k$ will be encoded by the form instance that consists of the root node and the nodes $n(a_1), \ldots, n(a_k)$. We denote this instance $I(a_1, \ldots, a_k)$. Hence, the initial state will be the instance $I(v_1, \ldots, v_k)$. The nodes $n(t)$, $t \in T$ will be used to control the transitions. For ease of notation, we introduce the notation *conf* for the formula $\neg(\bigvee_{t \in T} n(t))$, which is true if the instance does not contain any of the control nodes.

An end configuration is one that has no successors. That is, if there does not exist a transition $t \in T$ with $t = ((a, b), (c, d))$ such that a and c are both in the configuration. Hence, we can describe such a deadlock situation with the following completion formula:

$$\phi \;=\; conf \wedge \bigwedge_{((a,b),(c,d)) \in T} \neg(a \wedge c) \;\; .$$

In order to perform an actual transition $t = ((a, b), (c, d))$ in our form completability problem, we will use the control node $n(t)$ to denote that the transition is taking place. The addition condition of the control node $n(t)$ is thus $(conf \wedge n(a) \wedge n(c))$, and the deletion condition $(\neg n(a) \wedge \neg n(c) \wedge n(b) \wedge n(d))$. Furthermore, for every vertex v, the addition condition will be:

$$\neg v \wedge (\bigvee_{t=((x,v),(y,z)) \in T} n(t) \vee \bigvee_{t=((x,y),(z,v)) \in T} n(t)) \;\; ,$$

and the deletion condition:

$$(\bigvee_{t=((v,x),(y,z))\in T} n(t) \vee \bigvee_{t=((x,y),(v,z))\in T} n(t)) \ .$$

There are no other access rights.

The form instance $I(v_1,\ldots,v_k)$ is completable if and only if there is a reachable deadlock. □

COROLLARY 4.7. *Deciding semi-soundness for the class $F(A^-,\varphi^-,1)$ is **PSPACE**-complete.*

PROOF. The proof of the upper bound is similar to that of Theorem 4.6 except that we guess a sequence of canonical instances such that from the last canonical instance the form is not completable. Note that by Theorem 4.6 this last requirement can indeed be verified in **PSPACE**.

The **PSPACE**-hardness is shown by reducing the completability problem for this fragment to the semi-soundness problem for the same fragment. From a guarded form G we construct a guarded form G' with the same initial instance and a slightly larger schema. The access rules of G' are similar to that of G but ensure that for all reachable instances I and I' of G it holds that I' is reachable from I in G' iff I' is reachable from I in G or I' is the initial instance. It then follows by Lemma 4.3 that G' is semi-sound iff G is completable. The construction of G' is detailed by the following points:

- In the schema we add the fields reset and build under $\mathtt{r}$. Informally these will indicate the phase in which the instance is deleted an in which the initial instance is rebuilt, respectively.
- All access rules from G are copied except that for additions the formula ψ is transformed to $\psi\vee\neg\mathsf{reset}\vee\neg\mathsf{build}$ and for deletions it is translated to $\psi \vee \mathsf{reset}$.
- The completion formula φ is transformed to $\varphi\wedge\neg\mathsf{reset}\wedge\neg\mathsf{build}$.
- $A(\mathsf{add},\mathsf{reset}) = \neg\mathsf{build} \wedge \neg\mathsf{reset}$, $A(\mathsf{del},\mathsf{reset}) = \mathsf{build}$, $A(\mathsf{add},\mathsf{build}) = \mathsf{reset} \wedge \neg\mathsf{build} \wedge \neg(l_1 \vee \ldots \vee l_n)$ where $l_1,\ldots,l_n$ are the fields in the schema under the root, and $A(\mathsf{del},\mathsf{build})$ is a formula that tests if the instance is $can(I_0)$.
- Finally the access rules for additions are extended such that while build holds $can(I_0)$ is built.

□

4.2 Positive completion formulas

All the preceding results for fragments with unrestricted access rules and unrestricted completion formulas also hold for the fragments with only positive completion formulas. This is because we can simulate the unrestricted fragments as follows. We add in the schema a new field final under the root $\mathtt{r}$, let the completion formula be final and add access rules for final such that it can be added if the old completion formula holds. It is clear that the resulting guarded form has the completability and semi-soundness properties iff the original guarded form has them.

5. POSITIVE ACCESS RULES

5.1 Unrestricted completion formulas

THEOREM 5.1. *Deciding completability for guarded forms in the class $F(A^+,\varphi^-,k)$ is **NP**-hard, and for the class $F(A^+,\varphi^-,\infty)$ is **PSPACE**-hard.*

PROOF. Let φ be an arbitrary Boolean formula. We reduce deciding satisfiability of this formula to the completability problem of a guarded form as follows: for every variable x in φ, there is one node labelled x in the schema of the guarded form. All access rules are set to *true*. The completion formula is the given formula φ. It is easy to see that the guarded form is completable if and only if φ is satisfiable, because the access rules allow any instance that satisfies the schema to be constructed. □

THEOREM 5.2. *Deciding the completability of an instance I of a guarded form in $F(A^+,\varphi^-,k)$ is in **NP**.*

PROOF. (Sketch) Suppose that the guarded form is completable by the instance J, and let $o_1,\ldots,o_n$ be the path from the initial instance I to J. Then, we can show that if a deletion is followed by an addition we can swap the two and again obtain a possible path from I to J. So we can assume that there is a path between I and J that consists of a sequence $a_1,\ldots,a_k$ of additions, followed by a sequence $d_1,\ldots,d_{n-k}$ of deletions. In what follows we show that with only a polynomially large subsequence of this sequence we can also reach an instance that satisfies the completion formula φ.

Because of Lemma 4.4, within the instance J, we can identify a subtree T that satisfies φ, with branching factor linear in the length of φ. Because the depth of the guarded form is constant, the size of T is polynomial in the length of the completion formula φ. The whole construction that follows is aimed at constructing a minimal reachable instance that contains the tree T as a subtree. We call this tree the *witness tree.* During its construction we maintain two sets *Add* and *Del* that contain subsets of the additions $a_1 \ldots a_k$ and deletions $d_1,\ldots,d_{n-k}$, respectively, and we start with *Add* contain the addition that construct T and *Del* empty. Then we add additions and deletions to *Add* and *Del* if they are required by the access rules those that are already in these sets or by the completion formula. We can do this in such a way that in order to ensure that a certain formula ψ holds for a certain node we add at most one addition that adds a child under that node. Since all such formulas ψ will be subformulas or negations of subformulas in the guarded from it follows that the fan-out of the nodes in the trees that are added by the additions in the resulting *Add* set is linear in the size of the guarded form. Since the height of these trees is at most k and their number is at most the number of nodes in T it follows that *Add* is polynomial in the size of the guarded form. □

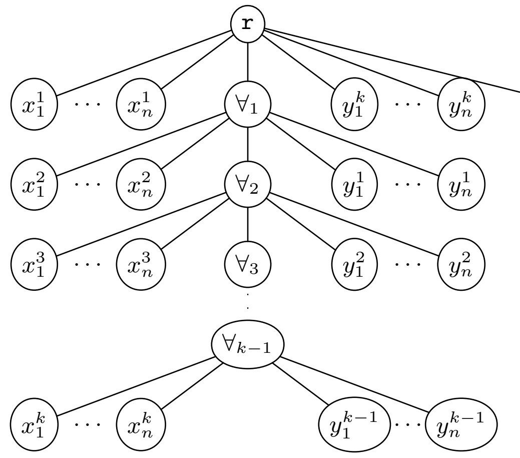
THEOREM 5.3. *Deciding the semi-soundness of guarded forms in $F(A^+,\varphi^-,k)$ is $\mathbf{\Pi^P_{2k}}$-hard.*

PROOF. We show $\mathbf{\Pi^P_{2k}}$-hardness by reducing the $QSAT_{2k}$-problem to the complement of semi-soundness. That is, given an instance of the $QSAT_{2k}$-problem, we construct a guarded form of $F(A^+,\varphi^-,k)$, that is *not* semi-sound if and only if the $QSAT_{2k}$-instance evaluates to true.

Consider the following $QSAT_{2k}$-instance:

$$\exists x^1_1 \ldots \exists x^1_{n_1} \forall y^1_1 \ldots \forall y^1_{m_1} \exists x^2_1 \ldots \exists x^2_{n_2} \forall y^2_2 \ldots \forall y^2_{m_2} \\ \ldots \exists x^k_1 \ldots \exists x^k_{n_k} \forall y^k_1 \ldots \forall y^k_{m_k} \psi \ .$$

We can assume without loss of generality that all quantifier blocks have the same number of variables; that is, $n_1 = \ldots = n_k = m_1 = \ldots = m_k = n$. The guarded form we construct for this $QSAT_{2k}$-instance, will have the following form schema:



The access rules are relatively simple: for all nodes, except $y_1^k, \ldots, y_n^k$ and uc, the addition and deletion conditions are: $\mathtt{r}/uc$; that is, they can be added and deleted, as long as the uc-element is present (uc stands for "under construction"). The access rules for addition and deletion of $y_1^k, \ldots, y_n^k$ are always true. The addition condition for uc is uc; that is, uc can only be added if it is already present. Put otherwise, if an instance no longer contains uc, then none of the instances reachable from this instance will contain uc. The deletion condition for uc is always true. To summarize: as long as uc is present, all elements can freely be added and deleted. Once uc is deleted, only manipulations of y_1^k, …, y_n^k are allowed. The initial instance of the guarded form is the instance consisting of only the root and one node labelled uc.

We will now construct the completion formula in such a way that there exists a reachable instance J that cannot be completed if and only if the $QSAT_{2k}$-instance evaluates to true. Intuitively, this instance J will represent a situation where for the first existential block, a choice has been made. For every possible assignment in the first universal block, there is an element labelled $\forall_1$ that represents this assignment. Within every assignment block $\forall_1$, a choice for the second existential block is made. For every $\forall_1$-element, for every possible assignment for the variables in the second universal block, an $\forall_2$-element representing this assignment is present, and so on, until the k-th existential block. Notice that there are no elements with label $\forall_k$ to represent the last universal quantifier block.

The completion formula is then defined as:

$$uc$$
$$\vee \left(\bigvee_{i=1}^{k-1} \forall_1/\ldots/\forall_{i-1}[\neg\forall_i[(\eta_1^i \wedge \ldots \wedge \eta_n^i]]\right)$$
$$\vee\ \forall_1/\ldots/\forall_{k-1}[\neg\psi']$$

where $\eta_j^i = y_j^i \leftrightarrow \mathtt{r}/y_j^k$ and ψ' is the following formula: all variables x_j^i are replaced by the path expression:

$$\overbrace{../\ldots/..}^{(k-i)\text{ times}}\ /x_j^i$$

all variables y_j^i, $i \neq k$, are replaced by:

$$\overbrace{../\ldots/..}^{(k-1-i)\text{ times}}\ /y_j^i$$

and the variables y_j^k are replaced by:

$$\overbrace{../\ldots/..}^{(k-1)\text{ times}}\ /y_j^k$$

The completion formula is of the form $uc \vee \ldots$ so a reachable incompletable instance does not contain uc.

Suppose that there exists an element $\forall_{i-1}$, such that there exists an assignment v that is not represented by a child element $\forall_i$. Then, we can complete the instance J by making the following disjunct in the completion formula true:

$$\forall_1/\ldots/\forall_{i-1}[\neg\forall_i[(y_1^i \leftrightarrow \mathtt{r}/y_1^k) \wedge \ldots \wedge (y_n^i \leftrightarrow \mathtt{r}/y_n^k)]]$$

Finally, suppose that the $QSAT_{2k}$ formula is true. Then we can find an assignment for x_1^1, …, x_n^1, such that for all assignments for y_1^1, …, y_n^1 there exists an assignment for x_1^2, …, x_n^2, such that for all ..., et cetera, ψ holds. We construct the instance that corresponds to these assignments. That is, we represent the assignment for x_1^1, …, x_n^1 under $\mathtt{r}$, for each assignment for y_1^1, …, y_n^1 we introduce an $\forall_1$ node that represents this assignment. Under this $\forall_1$ node we also add the corresponding assignment for x_1^2, …, x_n^2 and for each corresponding assignment for y_1^1, …, y_n^1 we introduce an $\forall_2$ node that represents this assignment. Et cetera, until we have added the nodes for the assignment for x_1^k, …, x_n^k under the $\forall_{k-1}$ nodes. Note that if this instance is reached only manipulations of y_1^k, …, y_n^k are allowed since there is no uc field. Moreover, from this instance the form cannot be completed since (1) the uc wil remain absent, (2) the second part of the completion formula cannot be satisfied because there are nodes for all possible assignments and (3) the third part cannot be satisfied because the $QSAT_{2k}$ formula is true. On the other hand, assume that the $QSAT_{2k}$ formula is not true. Then from every reachable instance the form can be completed since at least one of the following holds: (1) uc has not yet been removed, or (2) not at all levels i all assignments for the x_1^i, …, x_n^i have corresponding $\forall_i$ nodes that represent them, or (3) there is an assignment for y_1^k, …, y_n^k such that when encoded in the corresponding fields ensures that the instance satisfies $\forall_1/\ldots/\forall_{k-1}[\neg\psi']$. $\square$

Corollary 5.4. *Deciding the semi-soundness of guarded forms in the class $F(A^+, \varphi^-, \infty)$ is* **PSPACE**-*hard.*

Proof. We can reduce any $QSAT_{2k}$-instance to deciding semi-soundness for guarded forms of depth k. The proof in the previous proof was constructive, and uniform for different k's. Hence, $QSAT$ can be reduced to deciding semi-soundness for $F(A^+, \varphi^-, \infty)$. $\square$

5.2 Positive completion formulas

Theorem 5.5. *Deciding the completability of a guarded form in $F(A^+, \varphi^+, \infty)$ is in* **P**.

Proof. (Sketch) The completability of a guarded from in $F(A^+, \varphi^+, \infty)$ can be verified by adding as much edge as possible without adding a sibling with the same label as an already existing sibling. The result will satisfy φ iff the guarded form is completable. The only-if part is proved by showing that a sequence of additions that leads from the initial instance to an instance that satisfies φ can be transformed to a sequence additions that never adds a second sibling with the same label by removing such additions and replacing subsequent additions under the second sibling with similar additions under the first sibling. Since the intermediate instances cannot become larger than the product of the size of the initial instance and the size schema this can be done in polynomial time. $\square$

THEOREM 5.6. *Deciding semi-soundness of guarded forms in the class $F(A^+, \varphi^+, 1)$ is* **coNP***-hard.*

PROOF. (Sketch) We give a reduction of the **NP**-complete SAT problem to the problem of deciding whether a guarded form is not semi-sound. This shows that deciding semi-soundness is **coNP**-hard.

Let ψ be a Boolean formula in 3-conjunctive normal form, and let $x_1, \ldots, x_k$ be the variables in ψ. For every variable x_i, $i = 1 \ldots k$, we introduce two nodes, labelled respectively x_i and $\overline{x_i}$, in the schema of the guarded form. The presence of nodes with these labels represent respectively x_i is true and x_i is false. The initial form instance I in the semi-soundness problem consists of the root-node with all x_i and $\overline{x_i}$, $i = 1 \ldots k$. The access rules are as follows: $A(\mathsf{del}, x_i) = \overline{x_i}$, $A(\mathsf{del}, \overline{x_i}) = x_i$, $A(\mathsf{add}, x_i) = x_i$, and $A(\mathsf{add}, \overline{x_i}) = \overline{x_i}$. Finally, the acceptance condition φ will reflect the negation of the SAT formula ψ, and is defined as $neg(\psi)$, with $neg(\psi)$ recursively defined as follows: $neg(x_i) = \overline{x_i}$, and $neg(\neg x_i) = x_i$. Furthermore, for every conjunct of ψ, $neg(\ell_1 \vee \ell_2 \vee \ell_3)$ is defined as $neg(\ell_1) \wedge neg(\ell_2) \wedge neg(\ell_3)$. Finally, $neg(C_1 \wedge C_2 \wedge \ldots C_m)$ is defined as $neg(C_1) \vee neg(C_2) \vee \ldots \vee neg(C_m)$. It can then be shown that ψ is satisfiable iff the constructed guarded form is not semi-sound. $\square$

COROLLARY 5.7. *Deciding semi-soundness of a guarded form in the class $F(A^+, \varphi^+, 1)$ is* **coNP***-complete.*

PROOF. Because of Lemma 4.3, to show that a guarded form in $F(A^+, \varphi^+, 1)$ is not semi-sound, it suffices to give a canonical instance J, such that J cannot be reached from $can(I)$, and J is incompletable. Thus, we can guess such a canonical instance J. Because of the depth of 1, J is succinct. Both checking the reachability of J from $can(I)$ and the incompletability of J can be done in polynomial time. (Theorem 5.5) $\square$

Table 1: Summary of the complexity results.

Fragment	Completability	Semi-Soundness
$A^+, \varphi^+, 1$	**P**	**coNP**-compl.
A^+, φ^+, k	**P**	**coNP**-$\underline{\text{hard}}$
A^+, φ^+, ∞	**P**	**coNP**-$\underline{\text{hard}}$
$A^+, \varphi^-, 1$	**NP**-compl.	$\mathbf{\Pi_2^P}$-compl.
A^+, φ^-, k	**NP**-compl.	$\mathbf{\Pi_{2k}^P}$-$\underline{\text{hard}}$
A^+, φ^-, ∞	**PSPACE**-$\underline{\text{hard}}$	**PSPACE**-$\underline{\text{hard}}$
$A^-, \varphi^-, 1$	**PSPACE**-compl.	**PSPACE**-compl.
A^-, φ^-, k	undecidable	undecidable
A^-, φ^-, ∞	undecidable	undecidable
$A^-, \varphi^+, 1$	**PSPACE**-compl.	**PSPACE**-compl.
A^-, φ^+, k	undecidable	undecidable
A^-, φ^+, ∞	undecidable	undecidable

6. SUMMARY AND CONCLUSION

In this paper we have analyzed the complexity of deciding correctness properties, viz. completability and semi-soundness, for workflows that are implicitly defined by a form and instance-dependent access rules that determine which updates are allowed on the contents of the form. The results in this paper are summarized in Table 1. The underlined results indicate open problems since there an upper bound of the problems has not been determined yet.

In future research we intend to investigate the presented formalism as a mechanism for expressing workflows and see under which restrictions certain workflows can or cannot be expressed.

Acknowledgment

We would like to thank *Ron Addie*, *Hua Wang*, and *Richard Watson* from the University of Southern Queensland for participating in valuable early discussions about form-based web information systems that inspired the work discussed in this paper.

7. REFERENCES

[1] *Microsoft Office InfoPath*, 2003, `http://www.microsoft.com/office/infopath/prodinfo/default.mspx`.

[2] J. Boyer, D. Landwehr, R. Merrick, T. Raman, M. Dubinko, and L. Klotz, *XForms 1.0 (second edition)*, W3C Recommendation, March 2006, `http://www.w3.org/MarkUp/Forms/`.

[3] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, *Logical foundations of peer-to-peer data integration*, PODS, 2004, pp. 241–251.

[4] S. Dekeyser, J. Hidders, R. Watson, and R. Addie, *Peer-to-peer form based web information systems*, ADC 2006 (Hobart, Tasmania, Australia) (Gill Dobbie and James Bailey, eds.), Australian Computer Society, Inc., January 2006.

[5] Irini Fundulaki and Maarten Marx, *Specifying access control policies for XML documents with XPath*, SACMAT 2004 (Yorktown Heights, New York, USA), ACM Press, 2004, pp. 61–69.

[6] J. Hidders, M. Dumas, W. van der Aalst, A. ter Hofstede, and J. Verelst, *When are two workflows the same?*, CATS, 2005, pp. 3–11.

[7] Hajo A. Reijers, Selma Limam, and Wil M. P. van der Aalst, *Product-based workflow design*, Journal of Management Information Systems **20** (2003), no. 1, 229–262.

[8] Hans-Jörg Schek and Marc H. Scholl, *The relational model with relation-valued attributes*, Inf. Syst. **11** (1986), no. 2, 137–147.

[9] W. van der Aalst, *The application of petri nets to workflow management*, Journal of Circuits, Systems, and Computers **8** (1998), no. 1, 21–66.

[10] W. van der Aalst, A. Hirnschall, and H. Verbeek, *An alternative way to analyze workflow graphs*, CAiSE, 2002, pp. 535–552.