Merging Uncertain Multi-Version XML Documents

M. Lamine Ba
Institut Mines–Télécom;
Télécom ParisTech; LTCI
Paris, France
mouhamadou.ba@
telecom-paristech.fr

Talel Abdessalem
Institut Mines–Télécom;
Télécom ParisTech; LTCI
Paris, France
talel.abdessalem@
telecom-paristech.fr

Pierre Senellart
Institut Mines–Télécom;
Télécom ParisTech; LTCI
Paris, France
pierre.senellart@
telecom-paristech.fr

ABSTRACT

Merging is a fundamental operation in revision control systems that enables integrating different changes made to the same documents. In open platforms, such as Wikipedia, uncertainty is ubiquitous, essentially due to a lack of knowledge about the reliability of contributors. We propose in [2] a version control framework designed for uncertain multi-version tree-structured documents, based on a probabilistic XML model. In this paper, we define a merge operation that complements our framework and enables the conciliation of uncertain versions. We devise an efficient algorithm that implements the merge operation and prove its correction.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data models*; I.7.1 [**Document and Text Processing**]: Document and Text Editing—*Version control*

Keywords

XML, collaborative work, uncertain version control, merge

1. INTRODUCTION

Uncertain version control. Version control of uncertain data has concrete applicability in open environments such as web-scale editing platforms. Most of these platforms, in particular Wikipedia [1], are facing (1) the rapid growth of the number of contributors with different level of reliability, and (2) the will to provide the users with the most trustworthy content. This latter purpose is especially challenging because of uncertainties in data inherent to unreliable contributors, recurrent conflicts (contradictions or edit wars) and frequent malicious contributions (e.g., spam). Besides that, trust is a subjective notion which sorely depends on the user preferences.

So far, within web-scale collaborative platforms like Wikipedia, version control allows maintaining the integrity of each document by tracking all contributions, as well as their history and their authors. This gives therefore the ability to revert to a given revision when some editing problems such as vandalism acts and unsourced information appear. Used version control approaches are however not necessarily intended to the versioning of uncertain data, and they do not integrate this important property (the fact that the quality and trust in every single update operation varies) in their model. To tackle this lack, we propose in [2] an XML version control system tailored for uncertain tree-structured multi-version documents. The targeted applications mostly handle tree-structured data, or formats akin to it like XML documents: HTML or XHTML documents, office documents and structured Wiki formats. Our uncertain multi-version XML model is based on the general framework for updating probabilistic XML data proposed in [8]. This framework specifies uncertainty modeling and assessing in a typical version control process, ensuring the efficiency of updates.

Problem statement. The current paper extends our uncertain version control model with merge capabilities. Merging is a fundamental operation in version control systems. It allows integrating different changes (revisions) made to the same document. This operation is particularly helpful for software configuration management, where the configurations and their components can be built based on a combination of different versions of different elementary parts of the software (see [5]). In web-scale collaborative platforms, the merging operation as known in traditional version control systems is not yet supported. Its use in this kind of environment (large-scale, open and collaborative) is of interest, as soon as concurrent editing and alternative revisions are allowed. We detail motivations next.

Motivations. Amongst the motivations for this work, we can cite the following ones. On one hand, users may trust only some contributors and want to see the document resulting from the conciliation of their contributions, i.e., the merge of the revisions produced by these contributors. If the user preferences are known (e.g., based on her personal settings or past behaviour), a recommender system can be built on Wikipedia in order to propose to the user a version resulting from the merge of the contributions of her trusted authors. On the other hand, open platforms such as Wikipedia requires as a core functionality the merge of articles which overlap (articles related to the same topic or sharing a large common part). This operation is currently done manually and requires a lot of time and coordination between contributors. This results in a tedious and error-prone supervised merge process. Providing an automated integration processes of these articles is certainly useful. To this goal, a merging operation is needed and it has to take into account the uncertainty associated to the merged data. In this paper, we present our uncertain version control model with a merging operation that covers common deterministic merge scenarios over XML documents while managing uncertain data. We devise an efficient algorithm for merging uncertain multi-version XML documents and prove its correctness.

Outline. First, we present in Section 2 the merge and edit detection techniques used for XML documents. Then, we summarize in Section 3 our uncertain muti-version model. Section 4 concretely

[1] `http://www.wikipedia.org/`

DChanges 2013, September 10th, 2013, Florence, Italy.
ceur-ws.org Volume 1008, http://ceur-ws.org/Vol-1008/paper1.pdf .

presents our merging operation, as well as a corresponding efficient algorithm. Finally, we conclude the paper in Section 5.

2. XML MERGE ALGORITHMS

The increasing use of XML-based systems, in particular those with a built-in version control engine, has lead to the adoption of new XML merge techniques, e.g., [6, 10, 13]. These algorithms, aware of the tree-like structure of XML documents, have arisen as a reliable alternative to classical methods within XML settings. Indeed, traditional methods for merging text or binary files, cannot detect meaningfully the semantics of changes over trees. Most of current XML merge algorithms share as a baseline the diff step (edit detection) always preceding the generation of the merged document. Some main differences can be stated as follows: (i) two-way versus three-way approaches, that is, the use or not of the common base document from which merged ones are derived; (ii) the set of handled edit operations; (iii) the compliance to ordered XML elements or unordered ones and; (iv) the conflict management strategy. In the following, we briefly survey a few of these algorithms for merging XML documents. We refer to [3, 12] for a more exhaustive overview about deterministic XML merge and edit detection techniques.

Merging in [13] and [10] has tackled ordered XML trees, more suitable in some human-edited contexts such as structured reports, rich text formats, etc. In [13], the motivation was the synchronization of versions of documents edited by different users. The author has explored a structural two-way merge via a polynomial-time algorithm which directly computes two isomorphic trees representing the merge output from the two input XML documents. The trees are progressively built in a bottom-up fashion with nodes (having unique identifiers) from the two documents, while ensuring their isomorphism during this construction by applying a series of node insertion, deletion and update when a difference is detected. As a result, the process of generating isomorphic trees, thereby the merge result, slightly involves a detection of the differences between merged XML documents. Therefore, there is an implicit processing of edit changes. However, no details are given by the author about the processing of conflicts. As for [10], the focus was on the reintegration [2] of changes to a document in cases where multiple independently modified copies of the document have been made. The paper has proposed a three-way XML merging algorithm with clear merge rules (e.g., node sameness, node context) and a categorization of conflicts based on real-world use cases. In contrast to [13], the algorithm of Lindholm [10] uses a trees matching process detecting move operations in addition to insertions, deletions and updates of nodes. In its merge step, *core* and *optional* conflicts are defined: a core conflict (e.g., update/update of a node) will cause a failure of the merge, whereas an optional conflict (e.g., delete/update of a sub-tree) may be tolerated. The system does not pretend to resolve all conflicts, but it always reports unresolved scenarios. La Fontaine, in [6], has focused more on the best XML matching strategy regarding node insertions and deletions. An intermediate (optimal) XML diff file encoding the matches is used to ease the merge process with the help of an XML transformation language such as XSLT dialect. This algorithm was designed to run both in a two-way setting and a three-way one regardless of the considered XML document model. Note that the aforementioned XML merge algorithms are all deterministic.

In contrast, two-way merging operations in [11] and [1] are intended for uncertain XML documents. The followed process consists of the same steps as in deterministic settings. The main distinction with [11] is that its merge outcome is an XML document where nodes come with some elements modeling their amount of uncertainty (the synchronizer [11] is based on *Dempster–Shafer theory* to deal with uncertainty, in the form of probability values, degrees of beliefs, or necessity measures, associated to data) that does not retain enough information for retrieving back individual versions merged. [1] is most closely related in spirit to the current paper since both rely on the same general framework for managing uncertain XML in a typical versioning process; merging is not formally considered in [1].

[2]Merging changes that led to two distinct documents and apply the merge result into a third document.

3. UNCERTAIN MULTI-VERSION XML

A multi-version (XML) document with uncertain data evolves, through uncertain updates, and leads to uncertain versions. It is represented by the means of an *uncertain multi-version XML document model*, that describes the *version space* of this document together with a *probability distribution* over the set of possible versions.

The following is a concise summary of the formal foundations of our model and the evaluation of updates over it. For more details, see [2].

Model: Formal Definition. A multi-version XML document $\mathscr{T}_{mv}$ with uncertainty is a couple $(\mathscr{G},\Omega)$ where $\mathscr{G}$ is a directed acyclic graph (DAG) over a set $\mathscr{V}\cup\{e_0\}$ of events $e_0\dots e_n$ representing the version space of $\mathscr{T}_{mv}$, and Ω is a function giving the possible versions of the document, as we now detail.

An *event* e_i in $\mathscr{V}$ has a random nature and happens with a certain probability. It is defined as a conjunction of random Boolean variables $b_1\dots b_m$ that each model a given source of uncertainty (e.g., the source of information). This definition of the events using Boolean variables lies on the following: (i) variables are pairwise independent, that is, $Pr(b_j\wedge b_k)=Pr(b_j)\times Pr(b_k)$ for all $b_j\neq b_k$; (ii) a variable b_j, correlating different events, can be reused across events; (iii) one *revision variable* $b^{(i)}$, representing more specifically the uncertainty in the content, is not shared across other events and only occurs in e_i. Our version control system is *state-based* with events modeling the different uncertain states of the evolution of the versioned document. A state, i.e., an event, has contextual information about a given version (in the form of, first, Boolean random variables involved; second, an edit script Δ_i; third, possible other metadata).

The DAG $\mathscr{G}=(\mathscr{V}\cup\{e_0\},\mathscr{E})$ keeps the history of the evolution of $\mathscr{T}_{mv}$ with: (i) the particular event $e_0\notin\mathscr{V}$, which represents the initial state of $\mathscr{T}_{mv}$, as the root of $\mathscr{G}$; (ii) $\mathscr{E}\subseteq\mathscr{V}^2$ defines the set of directed edges of $\mathscr{G}$ that enable to implicitly track derivation relationships between the generated (uncertain) versions. A *branch* of $\mathscr{G}$ is a directed path in which the tail e_j is reachable from the head e_i by traversing a set of ordered edges in $\mathscr{E}$. A *rooted branch* is a branch with e_0 as head node.

A *version* of $\mathscr{T}_{mv}$ is an XML document mapping to a set of events in $\mathscr{G}$, the events whose edit scripts together made this version happen. Such an event set is always a rooted branch in $\mathscr{G}$ in a deterministic versioning case, whereas it can be arbitrary in the uncertain setting. In the model, formally an XML document is an *unordered*[3], *unranked*, and *labeled* tree $\mathscr{T}$ in which a node x has a unique identifier $\alpha(x)$ in $\mathscr{I}$ and a label $\phi(x)$ in $\mathscr{L}$ with $\mathscr{I}\cap\mathscr{L}=\emptyset$ (for brevity, we do not mention node identifiers when depicting example trees). In addition, all trees considered share the same root node (same label, same identifier). Given the set $2^{\mathscr{V}}$ of all subsets of $\mathscr{V}$ and the infinite set $\mathscr{D}$ of all XML trees, the mapping

[3]We leave the extension to ordered trees open as in [2].

$\Omega : 2^{\mathscr{V}} \to \mathscr{D}$ associates sets of events to versions of $\mathscr{T}_{mv}$ in such a way that (a) $\Omega(\emptyset)$ corresponds to the root-only XML tree of $\mathscr{D}$ and; (b) for all i, for all $\mathscr{F} \subseteq 2^{\mathscr{V}\setminus\{e_i\}}$, $\Omega(\{e_i\} \cup \mathscr{F}) = [\Omega(\mathscr{F})]^{\Delta_i}$ where Δ_i is the script attached to the event e_i and $[\Omega(\mathscr{F})]^{\Delta_i}$ its evaluation over the document $\Omega(\mathscr{F})$. A mapping Ω implicitly defines a probability distribution over the set of versions, as detailed in [2].

We have just defined an abstract multi-version XML document – we now provide a general and concise syntax for it, that has for semantics such a multi-version document.

Probabilistic Encoding: Syntax and Semantics. We have introduced in [2] a syntax $\widehat{\mathscr{T}_{mv}}$ for an uncertain multi-version XML document, based on probabilistic XML [9]. An *uncertain multi-version XML encoding* $\widehat{\mathscr{T}_{mv}}$ is defined by a pair $(\mathscr{G}, \widehat{\mathscr{P}})$ where (a) $\mathscr{G}$ is as before a DAG of events and; (b) $\widehat{\mathscr{P}}$ is a $\mathsf{PrXML}^{\mathsf{fie}}$ p-document with random variables $b_1 \dots b_m$ representing efficiently all possible versions and their corresponding event sets. Formally, the $\mathsf{PrXML}^{\mathsf{fie}}$ p-document $\widehat{\mathscr{P}}$ is an *unordered*, *unranked*, and *labeled* tree where every node (except the root) x may be annotated with an *arbitrary propositional formula* $\mathit{fie}(x)$ over $b_1 \dots b_m$. Different nodes in the p-document can be correlated by the use of common variables. A *valuation* ν of the variables $b_1 \dots b_m$ is a Boolean function that sets some variables to `true` and the remaining to `false`. This valuation ν produces over $\widehat{\mathscr{P}}$ the particular XML document $\nu(\widehat{\mathscr{P}})$, also known as a *possible world*, in which only nodes annotated with formulas valuated at `true` by ν are kept (nodes whose formulas are valuated to `false` by ν are deleted from the tree, along with their descendants). The probability of this document $\nu(\widehat{\mathscr{P}})$ is given by the sum of the probability of the valuations that yield the document. For a more detailed picture of the $\mathsf{PrXML}^{\mathsf{fie}}$ representation system, see [8, 9].

The semantics of an encoding $\widehat{\mathscr{T}_{mv}}$, denoted $[\![\widehat{\mathscr{T}_{mv}}]\!]$, is an uncertain multi-version XML document $(\mathscr{G}, \Omega)$. The DAG $\mathscr{G}$ does not change, whereas Ω is such that $\Omega(\mathscr{F}) := \nu_{\mathscr{F}}(\widehat{\mathscr{P}})$ for all $\mathscr{F} \subseteq \mathscr{V}$, where $\nu_{\mathscr{F}}$ is a valuation over variables $b_1 \dots b_m$ defined as follows. Let $B^+_{\mathscr{F}}$ be the set of all random variables occurring in one of the events of $\mathscr{F}$ and set $B^-_{\mathscr{F}}$ the set of all revision variables $b^{(i)}$'s for e_i *not in* $\mathscr{F}$. Then $\nu_{\mathscr{F}}$ sets variables of $B^+_{\mathscr{F}}$ to `true`, variables of $B^-_{\mathscr{F}}$ to `false`, and other variables to an arbitrary value. This semantics remains non-ambiguous as long as formulas occurring in $\widehat{\mathscr{P}}$ are expressed as formulas over the events of $\mathscr{V}$, i.e., do not make use of the Boolean variables separately of the events.

Updates: Semantics and Evaluation. An *edit script* Δ is a set of edit operation over XML nodes. An edit operation is either an insertion or a deletion of nodes. An *insertion* is formally defined as `ins(`i`,`x`)` with i the identifier of the node where the insertion must take place and x the label of the new node to be added. As for a *deletion*, it is introduced as `del(`i`)` where i represents the identifier of the node to remove. The evaluation of Δ over any XML document $\mathscr{T}$ produces the document $[\mathscr{T}]^{\Delta}$ by applying insertions and deletions to $\mathscr{T}$; if no node is selected by a given insertion or deletion, it is simply ignored.

An *update operation* is set up in the uncertain multi-version XML framework as $\mathtt{updOP}_{\Delta,\, e,\, e'}$ where Δ is an edit script, e is an actual event pointing to the edited version and e' is a fresh one assessing the uncertainty in this update. Its semantics on $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ (i) updates $\mathscr{G}$ to $(\mathscr{G} \cup (\{e'\}, \{(e, e')\})$ and; (ii) extends Ω to Ω' by letting for all $\mathscr{F} \subseteq \mathscr{V} \cup \{e'\}$: $\Omega'(\mathscr{F}) := \Omega(\mathscr{F})$ if $e' \notin \mathscr{F}$ and $\Omega'(\mathscr{F}) := [\Omega(\mathscr{F} \setminus \{e'\})]^{\Delta}$ otherwise. The translation of this semantics on the general syntax $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$ is done through an update algorithm `updPrXML` that first modifies $\mathscr{G}$ as before, and then it evaluates operations in Δ over the p-document $\widehat{\mathscr{P}}$ as follows. This is the usual implementation of updates in probabilistic XML [8] and we show that this is compatible with the semantics of multi-version encodings in [2].

- For an insertion $u =$ `ins(`i`,`x`)` in Δ: $\mathit{fie}(x)$ of x in $\widehat{\mathscr{P}}$ is set to $\mathit{fie}(x) \vee (e')$ if x already occurs in $\widehat{\mathscr{P}}$; otherwise, u inserts x in $\widehat{\mathscr{P}}$ with $\mathit{fie}(x) = e'$.
- For a deletion $u =$ `del(`i`)` in Δ: the node x in $\widehat{\mathscr{P}}$ such that $\alpha(x) = i$ (if it exists) has its formula $\mathit{fie}(x)$ updated to $\mathit{fie}(x) \wedge (\neg e')$.

EXAMPLE 3.1. *Figure 1 shows an uncertain multi-version document $\mathscr{T}_{mv}$ with: (a) the version space $\mathscr{G}$ with four staged events; (b) four event sets and associated versions; (c) the p-document $\widehat{\mathscr{P}}$ encoding all the possible versions based on staged events and their attached edit scripts, resulting in formulas attached to nodes (shown above each node). As a sketch, $(e_1 \wedge \neg e_2)$ reveals that* s_1 *was added at e_1 and then removed at e_2. The given four versions are exactly those modeled by deterministic systems. In contrast, the possible world mapping to $\{e_1, e_4\}$ is only valid within our framework. It occurs with the reject of the changes introduced by event e_2. Note that the probability of each possible version can be evaluated based on event sets that map to it and their probabilities.*

4. MERGE APPROACH

We detail in this section the translation of the usual XML merge operation within our uncertain versioning model.

A merge operation considers a set of versions and integrates their content in a single new one. We view this outcome as obtained via a three-way merge [4], that is, an integration of the changes from the inputs with respect to their common base version. We focus here on merging two versions which is the most common case in real applications. However, an extension to $n > 2$ versions is straightforward. In addition, we also assume that all the merged versions are only originated from updates over the base versions, i.e., we do not consider merging of versions with a different merge history – this is again for the sake of clarity of the exposition.

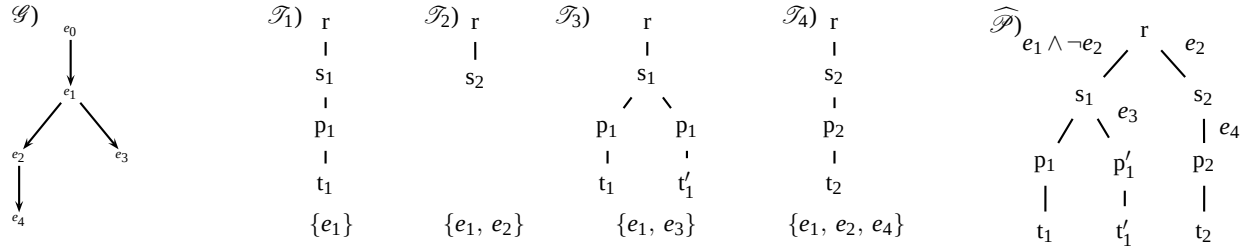
The merge process usually implies two steps: a) an extraction of the different sets of edit scripts that have led to the input versions and; b) a generation of the merge result by evaluating a unified set of the extracted edit scripts over the initial data. This last step must deal with possible conflicting edits (for the definition of conflicts, see next) due to concurrent changes (i.e., when two editors independently changes the same piece of data). The resolution of conflicts may yield several different content items for the merge. As a result, each possible merge outcome is obtained by making a choice between several possible edits. This naturally fits in the system with uncertainty handling because in such a setting there is no longer only one truth but several different possibilities, each with a certain probability of validity.

We first present the process of computing the edit scripts to use for the merge, as well as common merge scenarios. Then we introduce the semantics of merging uncertain multi-version XML documents, as well as an efficient algorithm on the probabilistic XML encoding.

4.1 Detection of Edits and Merge Scenarios

Assume an unordered XML document under version control. Let us consider two arbitrary versions $\mathscr{T}_1$ and $\mathscr{T}_2$, along with their common lowest ancestor $\mathscr{T}_a$, of this.

[4] A three-way merge enables a better matching of nodes and detection of conflicts.



(a) Version space (b) Four event sets and corresponding versions (c) PrXML$^{\text{fie}}$ p-document

Figure 1: Encoding of an uncertain multi-version XML document

4.1.1 Computation of Edit Scripts

We do not assume here given any explicit edit script. Instead of this, we include edit detection as an integral part of the merge process for the sake of generality. We define the edit script specifying the merge of versions $\mathscr{T}_1$ and $\mathscr{T}_2$ through the three-way diff algorithm *diff3*$(\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_a)$ on unordered trees with unique identifiers for nodes. The algorithm will return a script with only node inserts and deletes as edit operations. Like in [7], we set up our *diff3* based on the two-way diffs *diff2*$(\mathscr{T}_a, \mathscr{T}_1)$ and *diff2*$(\mathscr{T}_a, \mathscr{T}_2)$ as subroutines. These two-way functions separately compute two intermediate edit scripts using the same process.

- *diff2*$(\mathscr{T}_a, \mathscr{T}_1)$ initially matches the nodes in trees $\mathscr{T}_a$ and $\mathscr{T}_1$ in order to find out the shared, deleted, and inserted nodes. Then, the algorithm encodes the matches in terms of a set of node insertions and deletions which evaluated on $\mathscr{T}_a$ give $\mathscr{T}_1$. A node $x \in \mathscr{T}_a$ with no match in $\mathscr{T}_1$ is deleted, whereas a node $y \in \mathscr{T}_1$ with no match in $\mathscr{T}_a$ is added. Let us denote this edit script by Δ_1.
- *diff2*$(\mathscr{T}_a, \mathscr{T}_2)$ follows the same process and provides the script Δ_2 leading to $\mathscr{T}_2$ from $\mathscr{T}_a$.

A more global edit script, referred as Δ_3, models the final value of the *diff3*; Δ_3 is obtained by mixing Δ_1 and Δ_2. We describe this combination with three types of edits as follows.

Equivalent edits. An *equivalence* occurs between all edits in Δ_1 and Δ_2 with the same semantics and the same arguments (same identifiers and same labels). Specifically, two insertions $u_2 \in \Delta_1$ and $u_4 \in \Delta_2$ are equivalent if they specify the same node identifier and the same label to be added. As for deletions in Δ_1 and Δ_2, there is an equivalence between two if these target the same node. Given two equivalent edits, only one of the two operations is kept in Δ_3.

Conflicting edits. Any two given operations $u_2 \in \Delta_1$, $u_4 \in \Delta_2$ are *conflicting edits* when they come with different semantics, i.e, if u_2 is an insertion, then u_4 is a deletion (and conversely), and the insertion has added some new nodes as descendants of the node that is removed with the delete operation. We introduce conflicted edits in Δ_3 to be those satisfying the properties given above. Given that, we refer to the set of all conflicting edits in Δ_3 with $\Delta^{\mathscr{C}}$. We say that a node handled by conflicted edits is a *conflicted node*.

Independent edits. Those edits in Δ_1 and Δ_2 that do not belong to the two first classes. The set of equivalent and independent edits form the *non-conflicted* edits of a given diff algorithm. A node impacted by a non-conflicted edit is a *non-conflicted node* for a given merge operation. (Note that conflicted and non-conflicted nodes together form the set of all nodes impacted by edit scripts in Δ_3).

Now, let us briefly present the merging scenarios (cf. usual merge options, especially *mine-conflict* and *theirs-conflict*, in tools like *SubVersion* [4]) using *diff*s and *input versions*.

4.1.2 Deterministic Merge Scenarios

A large majority of current versioning models provide three common merge scenarios that consider the resolution of possible conflicts. Recall that in most cases, this resolution is manual, that is, it requires user involvement. Let $\mathscr{T}_m$ be the outcome of the merge of $\mathscr{T}_1$ and $\mathscr{T}_2$. We formalize the possible merge scenarios as follows.

1. First, one would like to perform the merge based on $\mathscr{T}_1$ and by updating this with the non-conflicted edits from Δ_2. For this case, we have $\mathscr{T}_m = [\mathscr{T}_1]^{\Delta_2 - \Delta^{\mathscr{C}}}$.
2. The second scenario is symmetric to the first one: it considers as a base version $\mathscr{T}_2$ and fetches the non-conflicted edits from Δ_1. For this case, we set $\mathscr{T}_m = [\mathscr{T}_2]^{\Delta_1 - \Delta^{\mathscr{C}}}$.
3. Finally, the last case maps to the update of the common version $\mathscr{T}_a$ with the non-conflicted edits in Δ_3, that is, one would like to reject all the conflicting edits in the merge outcome. For this case, we set $\mathscr{T}_m = [\mathscr{T}_a]^{\Delta_3 - \Delta^{\mathscr{C}}}$.

It is straightforward to show that when $\Delta^{\mathscr{C}} = \emptyset$, then we obtain the same content for the three merge scenarios. This observation is inherent to the computation of the edit scripts and the definition of the merge outcome in each scenario. Observe that we do not deal with the (intuitive and naive) merge case where the user corrects the conflicting parts with new inputs. However, this case can be simply treated by first choosing one the three outcome above and then by performing updates over this.

4.2 Merging Uncertain Multi-Version XML

We now introduce our abstraction of the merge operation (covering at least the set up of the merge scenarios above) within the uncertain multi-version XML document model.

For sure, an uncertain context induces an inherent uncertain merge; involved versions and *diff*s come with uncertainties. Let $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ be an uncertain multi-version XML document with n staged version control events. In addition, we consider $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$ as the probabilistic XML encoding of $\mathscr{T}_{mv}$. Recall again that each version of $\mathscr{T}_{mv}$ is identified with a particular event in $\mathscr{G}$, the one representing the tail of the branch of $\mathscr{G}$ leading to this version. We reason on events instead of full versions since these are here uncertain and can be defined in an arbitrary manner using events. This section introduces the formalism of the merge operation over any uncertain multi-version XML document and the mapping algorithm over its probabilistic XML encoding.

4.2.1 Abstracting Uncertain Merge Operation

With the help of the triple (e_1, e_2, e'), we refer in our setting with uncertainty to a merge operation as $\mathfrak{MRG}_{e_1,e_2,e'}$ where e_1 and e_2 point to the two versions to be merged and e' is a new event assessing the amount of uncertainty in the merge operation. We evaluate the semantics of such a merge operation over $\mathscr{T}_{mv}$ with uncertainty

as follows.

$$\mathfrak{MRG}_{e_1,e_2,e'}(\mathscr{T}_{mv}) := (\mathscr{G} \cup (\{e'\}, \{(e_1,e'),(e_2,e')\}),\ \Omega').$$

On the one hand, this evaluation inserts a new event and two edges in the version space $\mathscr{G}$. On the other hand, it generates a new distribution Ω' which represents an extension of Ω with new possible versions and event sets. Let $\mathscr{A}_{e_1}$ and $\mathscr{A}_{e_2}$ be the set of all strict *ancestor events* in $\mathscr{G}$ of e_1 and e_2 respectively. We denote the common set by $\mathscr{A}_s = \mathscr{A}_{e_1} \cap \mathscr{A}_{e_2}$. For all subset $\mathscr{F} \in 2^{\mathscr{V} \cup \{e'\}}$, formally we set:

- if $e' \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := \Omega(\mathscr{F})$;
- if $\{e_1,e_2,e'\} \subseteq \mathscr{F}$: $\Omega'(\mathscr{F}) := \Omega(\mathscr{F} \setminus \{e'\})$;
- if $\{e_1,e'\} \subseteq \mathscr{F}$ and $e_2 \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))]^{\Delta_2 - \Delta^{\mathscr{C}}}$;
- if $\{e_2,e'\} \subseteq \mathscr{F}$ and $e_1 \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_1} \setminus \mathscr{A}_s))]^{\Delta_1 - \Delta^{\mathscr{C}}}$;
- if $\{e_1,e_2\} \cap \mathscr{F} = \emptyset$ and $e' \in \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)))]^{\Delta_3 - \Delta^{\mathscr{C}}}$;

We consider the aforementioned edit scripts as all obtained via the *diff3* process sketched in Section 4.1.1. For each involved case, the *diff3* is executed on the (uncertain) arbitrary versions $\mathscr{T}_1 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_1}) \cup \{e_1\})$ and $\mathscr{T}_2 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_2}) \cup \{e_2\})$, and $\mathscr{T}_a = \Omega(\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_s)$ where $\mathscr{F}$ is the subset of events in $\mathscr{V} \cup \{e'\}$ considered as valid.

EXAMPLE 4.1. *Figure 2 describes the process of merging two possible versions, denoted by $\mathscr{T}_1$ and $\mathscr{T}_2$, from Figure 1 given their common base $\mathscr{T}_a$. In our proposal, this operation is simply encompassed with the merge specified over events e_3 and e_4 which point to the two input versions. On the left-hand side of the example, we provide the versions $\mathscr{T}_1$, $\mathscr{T}_2$ and $\mathscr{T}_a$ together with edit scripts $\{u_2,u_4\}$ and $\{u_3\}$ that led to them from the base $\mathscr{T}_a$. Typically, we view these scripts as given by diff functions outlined in Section 4.1.1 based on full versions. The right-hand side in Figure 2 explains the process of merging $\mathscr{T}_1$ and $\mathscr{T}_2$ (with the merge event e' evaluating the uncertainty in the merge) as follows: (i) First, all the edits in the scripts above coming with no conflicts, i.e., here only u_4 are validated for building the part of the merge (seen as an intermediate outcome) that is certain with the existence of e'; (ii) Then, generating the set of possible merge items by enumerating the different possibilities with the conflicting edits u_2 and u_3. The two initial possible results are obtained by propagating respectively u_2 and u_3 given the intermediate outcome. Such a propagation will give in the first case a merged version that only contains the sub-tree s_2, and in the second case a merged version with the sub-tree s_1 (including nodes p_1 and p'_1) in addition. Concretely, our merge approach will compute the same merged documents by first considering the input versions $\mathscr{T}_1$ and $\mathscr{T}_2$, and then by updating these with the edits without conflicts respectively from $\{u_3\}$ and $\{u_2,u_4\}$. Finally, the last possible content for the merge is obtained by discarding all the conflicting edits and by combining the concurrent nodes in the base version with the intermediate result.*

The uncertain merging operation as formalized above remains however intractable since it requires to evaluate every possible version for computing the overall merge result. Below, we propose a more convenient way to do this merge.

4.2.2 Merging over Probabilistic XML Encoding

We efficiently present the semantics of the merge operation in $\widehat{\mathscr{T}_{mv}}$ as Algorithm 1, namely `mergePrXML`. Prior to a deeper description of the proposed algorithm, we start by introducing the notion of *conflicted nodes* in the PrXML$^{\text{fie}}$ probabilistic encoding given the merge of events e_1 and e_2.

The history of edits over any specific node in $\widehat{\mathscr{P}}$ is encoded with its attached formula. We base on this for detecting the conflicted nodes. Let us set the following valuations of events in $\mathscr{G}$: (i) ν_s setting the events in $\mathscr{A}_s$ to true and the revision variables of all other events to false; (ii) ν_1 assigning a true value to the events in $\mathscr{A}_{e_1} \cup \{e_1\}$ and a false value to the revision variables of the other events and finally; (iii) ν_2 setting the events in $\mathscr{A}_{e_2} \cup \{e_2\}$ to true and all the revision variables in the remaining events to false.

We first introduce the lineage of an uncertain node in the PrXML$^{\text{fie}}$ p-document.

DEFINITION 4.1. *(Node lineage) The lineage formula of a given node $x \in \widehat{\mathscr{P}}$, denoted by $fie^{\uparrow}(x)$, is the propositional formula resulting from the conjunction of the formula of this node x with the formulas attached to all its ancestor nodes in $\widehat{\mathscr{P}}$.*

Instead of its formula[5], the lineage of a given node in the p-document encodes the entire history of edits, starting from the initial event, over the path leading to this node. Given that, we can approach the conflicted nodes in the p-document using their lineage formulas as follows.

DEFINITION 4.2. *(Conflicted node) Under the general syntax $\widehat{\mathscr{T}_{mv}}$, we say that a given x in $\widehat{\mathscr{P}}$ is a conflicted node with respect to the merge implying the events e_1 and e_2 when its lineage satisfies the following conditions:*

1. *$fie^{\uparrow}(x) \models \nu_s$;*
2. *$fie^{\uparrow}(x) \not\models \nu_1$ (or $fie^{\uparrow}(x) \not\models \nu_2$) and;*
3. *$\exists y \in \widehat{\mathscr{P}}$, `desc(x, y)`: $fie^{\uparrow}(y) \not\models \nu_s$ and $fie^{\uparrow}(y) \models \nu_2$ (or $fie^{\uparrow}(y) \models \nu_1$) where `desc(x, y)` means that y is a descendant of the node x.*

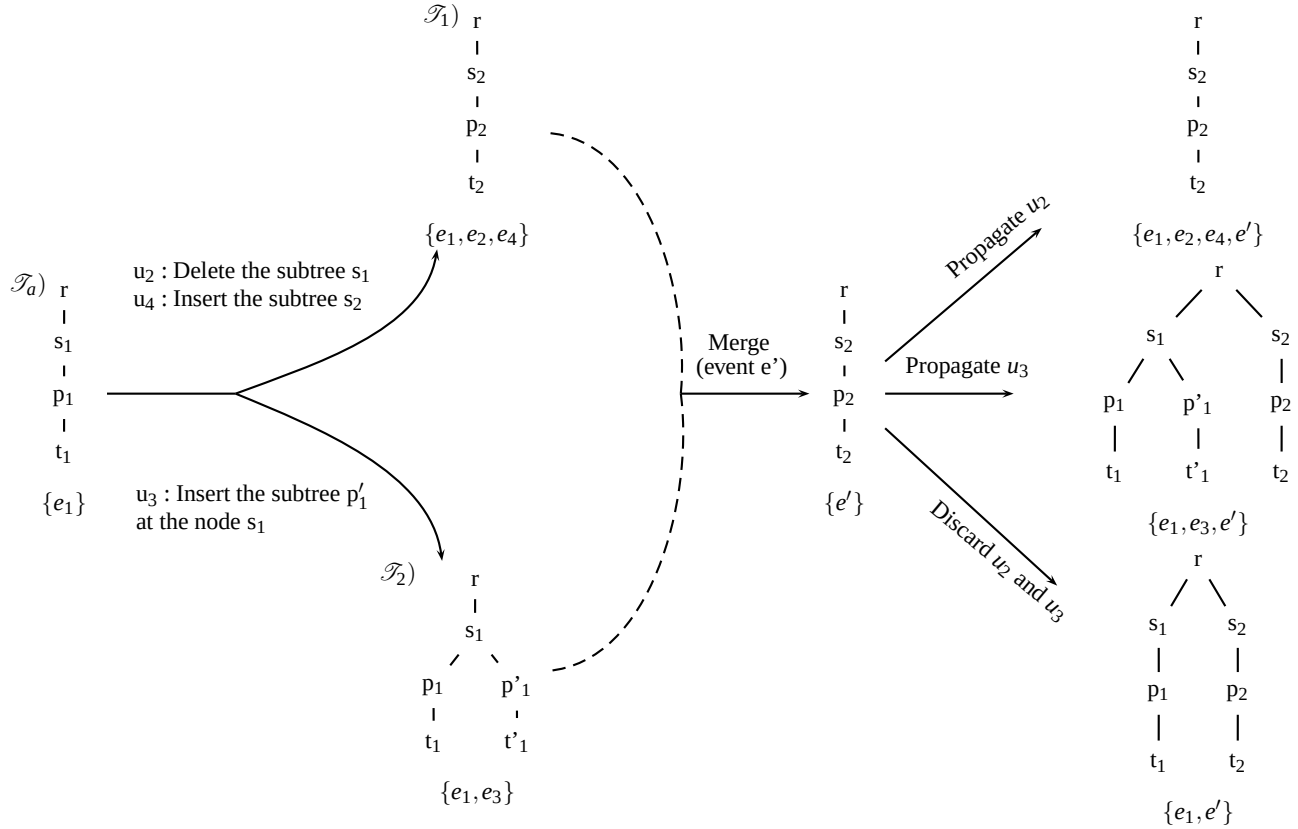
PROPOSITION 4.1. *Definition 4.2 is consistent with the definition of conflicted nodes given in Section 4.1.1.*

PROOF. (Sketch) Let x in $\widehat{\mathscr{P}}$ be a conflicted node such that 1) $fie^{\uparrow}(x) \models \nu_s$; 2) $fie^{\uparrow}(x) \not\models \nu_1$; 3) $fie^{\uparrow}(y) \not\models \nu_s$ and $fie^{\uparrow}(y) \models \nu_2$ with `desc(x, y)` true. The relation 1) yields $x \in \nu_s(\widehat{\mathscr{P}})$ which is a document corresponding to the common lowest ancestor of the versions $\nu_1(\widehat{\mathscr{P}})$ and $\nu_2(\widehat{\mathscr{P}})$. The relations 2) means that $x \notin \nu_1(\widehat{\mathscr{P}})$, i.e., in the history of edits that gave $\nu_1(\widehat{\mathscr{P}})$ from $\nu_s(\widehat{\mathscr{P}})$ there was at least a deletion u_2 over the node x. This is implied by the way `updPrXML()` proceeds. Besides that, 3) enables us to write in one side $x \in \nu_2(\widehat{\mathscr{P}})$ since $y \in \nu_2(\widehat{\mathscr{P}})$ and on another side $y \notin \nu_s(\widehat{\mathscr{P}})$. As a result, in the history of edits that led to $\nu_2(\widehat{\mathscr{P}})$ from $\nu_s(\widehat{\mathscr{P}})$ there was an insertion u_4 which added y as a child of x. In other words, u_2 and u_4 define two conflicted edits performed on the same node x. □

A conflicted node in $\widehat{\mathscr{P}}$ results in conflicting descendants. We refer to the conflicted set of nodes in $\widehat{\mathscr{P}}$ according to the merge of events e_1 and e_2 as the restriction $\widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\,e_2\}}}$. Under this, we infer below the non-conflicted set of nodes.

DEFINITION 4.3. *(Non-conflicted node) For the merge of events e_1 and e_2, we define a non-conflicted node x as a node in $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\,e_2\}}}$ having a formula $fie(x)$ satisfying one of the following conditions.*

[5]The formula just describes the semantics of edits from the event where the node was inserted for the fist time.



a) $\mathscr{T}_a$ (common base); $\mathscr{T}_1$ and $\mathscr{T}_2$ (versions to merge); $\{u_2, u_4\}$ and $\{u_3\}$ (edit scripts)

b) First, validating u_4 which does not have any conflict. Then, resolving the conflict between u_2 and u_3.

Figure 2: Merge Operation: (a) Input versions and (b) Generation of Merge results

1. $fie(x) \models v_s$, $fie(x) \not\models v_1$ *and* $fie(x) \not\models v_2$.
2. $fie(x) \not\models v_s$, $fie(x) \models v_1$ *and* $fie(x) \models v_2$.
3. $fie(x) \models v_s$, $fie(x) \models v_1$ *and* $fie(x) \not\models v_2$.
4. $fie(x) \models v_s$, $fie(x) \not\models v_1$ *and* $fie(x) \models v_2$.
5. $fie(x) \not\models v_s$, $fie(x) \models v_1$ *and* $fie(x) \not\models v_2$.
6. $fie(x) \not\models v_s$, $fie(x) \not\models v_1$ *and* $fie(x) \models v_2$.

PROPOSITION 4.2. *Definition 4.3 is consistent with the definition of non-conflicted nodes given in Section 4.1.1.*

The proof is straightforward. To be exhaustive about non-conflicted nodes, we infer the following lemma.

LEMMA 4.1. *Let us assume the merge over events e_1 and e_2. Given the sets $\mathscr{F}_s \subseteq \mathscr{A}_s$, $\mathscr{F}_1 \subseteq (\mathscr{A}_{e_1} \cup \{e_1\}) \setminus \mathscr{A}_s$ and $\mathscr{F}_2 \subseteq (\mathscr{A}_{e_2} \cup \{e_2\}) \setminus \mathscr{A}_s$, the expression of $fie(x)$ for any non-conflicted node $x \in \widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$ is consistent with one of the following formulas.*

1. $(\bigwedge_{e_i \in \mathscr{F}_s} (e_i)) \wedge \neg \left(\bigwedge_{e_j \in (\mathscr{F}_1 \cup \mathscr{F}_2)} (e_j) \right)$
2. $(\bigwedge_{e_i \in \mathscr{F}_1} (e_i)) \vee \left(\bigwedge_{e_j \in \mathscr{F}_2} (e_j) \right)$
3. $\left((\bigwedge_{e_i \in \mathscr{F}_s} (e_i)) \vee \left(\bigwedge_{e_j \in \mathscr{F}_1} (e_j) \right) \right) \wedge \neg (\bigwedge_{e_k \in \mathscr{F}_2} (e_k))$
4. $\left((\bigwedge_{e_i \in \mathscr{F}_s} (e_i)) \wedge \neg \left(\bigwedge_{e_j \in \mathscr{F}_1} (e_j) \right) \right) \vee (\bigwedge_{e_k \in \mathscr{F}_2} (e_k))$
5. $(\bigwedge_{e_i \in \mathscr{F}_1} (e_i))$
6. $(\bigwedge_{e_i \in \mathscr{F}_2} (e_i))$

PROOF. The proof relies on Definition 4.3. □

Let us continue this section by first describing `mergePrXML`, then by demonstrating its correctness with respect to the abstraction of the merge operation in Section 4.2.1.

Input: $(\mathscr{G}, \widehat{\mathscr{P}})$, e_1, e_2, e'
Output: Merging Uncertain XML Versions in $\widehat{\mathscr{T}_{mv}}$
$\mathscr{G} := \mathscr{G} \cup (\{e'\}, \{(e_1, e'), (e_2, e')\})$;
foreach *non-conflicted node x in* $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$ **do**
  `replace`($fie(x)$, e_1, $(e_1 \vee e')$);
  `replace`($fie(x)$, e_2, $(e_2 \vee e')$);
return $(\mathscr{G}, \widehat{\mathscr{P}})$

Algorithm 1: Merge Algorithm (`mergePrXML`)

Algorithm 1 considers as inputs the probabilistic encoding $(\mathscr{G}, \widehat{\mathscr{P}})$ of an uncertain multi-version XML document $\mathscr{T}_{mv}$, the events e_1 and e_2 of $\mathscr{G}$, and the new event e' modeling both the merge content items and the amount of uncertainty in these. Given that, `mergePrXML` first updates $\mathscr{G}$ as specified in Section 4.2.1. Then, the merge in $\widehat{\mathscr{P}}$ will result in a slight change in formulas attached to certain non-conflicting nodes in $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$. The function `replace` modifies such formulas by substituting all occurrences of e_1 and e_2 by $(e_1 \vee e')$ and $(e_2 \vee e')$ respectively. The idea is that each possible merge outcome, which occurs when e' is valuated to `true` regardless of the valuation of the other events, must come with at least the non-conflicted nodes from $\widehat{\mathscr{P}}$ seen as valid with e_1 and e_2. The remaining non-conflicted nodes, whose existence are independent of e_1 and e_2, will depend uniquely on the valuation of their ancestor events in each given valid event set including e'. At least, the validity of a conflicting node in a merge result relies on the *probability* of e_1 and e_2 when the event is e' *certain*. If e', together with e_1, are only valuated to `true`, we say that e_1 is more *probable*

than e_2 for the merge; in this case, only conflicted nodes valid with $\mathscr{A}_{e_1} \cup \{e_1\}$ are chosen. The converse works in the same manner. Any conflicted node will be rejected with a valuation setting e' to true and the revision variables in both e_1 and e_2 to false.

Assume an uncertain multi-version XML document $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ and the corresponding probabilistic XML encoding $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$. In addition, let us define $[\![.]\!]$ as the semantics operator which, applied on $\widehat{\mathscr{T}_{mv}}$, yields its correct semantics $[\![\widehat{\mathscr{T}_{mv}}]\!] = (\mathscr{G}, [\![\widehat{\mathscr{P}}]\!])$ such that $\mathscr{G}$ is the same as in $\mathscr{T}_{mv}$ and $[\![\widehat{\mathscr{P}}]\!]$ defines the same probability distribution over a subset of documents in $\mathscr{D}$ than Ω. Given a merge operation $\mathfrak{MRG}_{e_1,e_2,e'}$, we now show the main result of this paper:

PROPOSITION 4.3. *The definition of Algorithm 1 is correct with respect to the semantics of the merge operation over the uncertain multi-version XML document. In other words, the following diagram commutes:*

PROOF. Assume: $\begin{cases} \mathfrak{MRG}_{e_1,e_2,e'}([\![\widehat{\mathscr{T}_{mv}}]\!]) = (\mathscr{G}', \Omega') \\ \widehat{\mathscr{T}'_{mv}} = (\mathscr{G}', \widehat{\mathscr{P}'}) \text{ and } [\![\widehat{\mathscr{T}'_{mv}}]\!] = (\mathscr{G}', \Omega'') \end{cases}$

Seeing that we reach the same version space using mergePrXML is trivial. Now, we have to show that to Ω' will correspond $[\![\widehat{\mathscr{P}'}]\!]$; that is, $\Omega' = \Omega''$. Given each set $\mathscr{F} \subseteq \mathscr{V}'$, five scenarios must be checked for this equality.

1. For each subset $\mathscr{F}$ such that $e' \notin \mathscr{F}$, we have $\Omega'(\mathscr{F}) = \Omega(\mathscr{F})$. By definition, $\Omega(\mathscr{F}) = \nu(\widehat{\mathscr{P}})$ where ν is a valuation setting the special revision variable in e' to false and the other events to an arbitrary value. Abstracting out the formulas, we can claim that $\widehat{\mathscr{P}} \sim \widehat{\mathscr{P}'}$ regarding mergePrXML. Since $e' \not\models \nu$, the result of the evaluation of ν over $(e_1 \vee e')$ and $(e_2 \vee e')$ (or their negation) only depends on the truth values of e_1 and e_2 respectively. Thus by replacing in formulas of $\widehat{\mathscr{P}'}$ all occurrences of $(e_1 \vee e')$ and $(e_2 \vee e')$ by e_1 and e_2 respectively, we are sure to build a p-document $\widehat{\mathscr{P}''}$ with $\nu(\widehat{\mathscr{P}'}) = \nu(\widehat{\mathscr{P}''})$. But by the definition of mergePrXML, $\widehat{\mathscr{P}''}$ is exactly $\widehat{\mathscr{P}}$. As a result, we obtain $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$. Knowing beforehand that $\Omega''(\mathscr{F}) = \nu(\widehat{\mathscr{P}'})$, we can state that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ for any $\mathscr{F} \subseteq \mathscr{V}' \setminus \{e'\}$.

2. For each subset $\mathscr{F}$ such that $\{e_1, e_2, e'\} \cap \mathscr{F} \neq \emptyset$, we have $\Omega'(\mathscr{F}) = \Omega(\mathscr{F} \setminus \{e'\})$. Let ν be a valuation setting all the events in $\mathscr{F} \setminus \{e'\}$ to true, the revision variable in e' to an arbitrary value and the revision variables in the remaining events to false. Since e' does not occur in formulas in $\widehat{\mathscr{P}}$, we can write $\Omega(\mathscr{F} \setminus \{e'\}) = \nu(\widehat{\mathscr{P}})$ for sure. At this step, we resort to the logical consequences $(e_1 \models \nu) \Rightarrow ((e_1 \vee e') \models \nu)$ and $(e_2 \models \nu) \Rightarrow ((e_2 \vee e') \models \nu)$ regardless of the truth-value of the event e'. In the same way, $(\neg e_1 \not\models \nu) \Rightarrow (\neg(e_1 \vee e') \not\models \nu)$ and $(\neg e_2 \not\models \nu) \Rightarrow (\neg(e_2 \vee e') \not\models \nu)$. Therefore, by substituting in formulas of $\widehat{\mathscr{P}'}$ all occurrences of $(e_1 \vee e')$ and $(e_2 \vee e')$ by e_1 and e_2 respectively, we obtain the old p-document $\widehat{\mathscr{P}}$ with $\nu(\widehat{\mathscr{P}'}) = \nu(\widehat{\mathscr{P}})$ given the semantics of mergePrXML. Moreover, $\Omega'(\mathscr{F}) = \nu(\widehat{\mathscr{P}})$ because $\Omega'(\mathscr{F}) = \Omega(\mathscr{F} \setminus \{e'\})$ and $\Omega(\mathscr{F} \setminus \{e'\}) = \nu(\widehat{\mathscr{P}})$. So by inference, we can demonstrate that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ using the relations $\Omega'(\mathscr{F}) = \nu(\widehat{\mathscr{P}})$, $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$ and $\nu(\widehat{\mathscr{P}'}) = \Omega''(\mathscr{F})$ [6].

3. For each subset $\mathscr{F}$ such that $\{e_1, e'\} \cap \mathscr{F} \neq \emptyset$ and $e_2 \notin \mathscr{F}$, we have $\Omega'(\mathscr{F}) = [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))]^{\Delta_2 - \Delta^{\mathscr{C}}}$. Let $\mathscr{F}_1^+ = ((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))$ be the set of valid events excepting the valid ancestor events of e_2 in $\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)$. For the valuation ν setting all the events in $\mathscr{F}_1^+$ to true and the revision variables in the remaining events to false, Scenarios 1 and 2 enable us to write $\nu(\widehat{\mathscr{P}}) = \Omega(\mathscr{F}_1^+)$ and $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$. Now, we just need to show that $[\nu(\widehat{\mathscr{P}'})]^{\Delta_2 - \Delta^{\mathscr{C}}} = \nu'(\widehat{\mathscr{P}'})$ (ν' being the valuation that sets all the events in $\mathscr{F}$ to true and the revision variables in all others to false) to obtain the expected proof, that is, $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$. Let us set $\Delta = \Delta_2 - \Delta^{\mathscr{C}}$. We distinguish the two following cases.

(a) For the class of nodes x in $\nu(\widehat{\mathscr{P}'})$ unmodified by Δ. That is, $x \in \nu(\widehat{\mathscr{P}'})$ and $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$. For each node x in this class, $fie^{\uparrow}(x)$ in $\widehat{\mathscr{P}'}$ requires the trueness of events in $\mathscr{F}_1^+$ since $x \in \nu(\widehat{\mathscr{P}'})$. Moreover, by the definition of Δ, x may be either a conflicted node or its existence is independent of the values of events in $(\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\})$. If x is a conflicted node, it is intuitive to see that $fie^{\uparrow}(x)$ in $\widehat{\mathscr{P}'}$ is satisfied if events in $\mathscr{F}_1^+$ are all set to true and not if only those in $(\mathscr{F} \cap \mathscr{A}_{e_2})$ are set to true (cf. Definition 4.2.1 and updPrXML). In the another case, $fie^{\uparrow}(x)$ is only function of the values of events in $\mathscr{F}_1^+$ for the set $\mathscr{F}$ (typically when $fie^{\uparrow}(x)$ is Expression 5 in Lemma 4.1 with $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$). In both cases, we can state that $fie^{\uparrow}(x) \models \nu'$ since ν' similarly to ν sets all the events in $\mathscr{F}_1^+$ to true. As a result, we have $x \in \nu'(\widehat{\mathscr{P}'})$ when x belongs to this class.

(b) For the class of nodes $x \in \nu(\widehat{\mathscr{P}'})$ handled with Δ. Let $x \notin [\nu(\widehat{\mathscr{P}'})]^{\Delta}$, that is, there is an operation u in Δ that removes x from $\nu(\widehat{\mathscr{P}'})$. By the construction of Δ, it is easy to show that $fie(x)$ in $\widehat{\mathscr{P}}$ maps to Expression 3 in Lemma 4.1 with $\mathscr{F}_s = \mathscr{F}_1^+ \cap \mathscr{A}_s$, $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$ and $\mathscr{F}_2 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$. In $\widehat{\mathscr{P}'}$, this formula $fie(x)$ is just updated across Algorithm 1 by replacing the events e_1 (the disjunction) in the first member and e_2 in the second one respectively by $(e_1 \vee e')$ and $(e_2 \vee e')$. Since ν' sets all the events in $\mathscr{F}$ to true, therefore the first member of $fie(x)$ will be valuated to true while the second member will be valuated to false. As a consequence, clearly we can state that $fie(x) \not\models \nu'$, i.e., $x \notin \nu'(\widehat{\mathscr{P}'})$. In summary, we proven that for each node x in $\nu(\widehat{\mathscr{P}'})$ deleted with Δ, x is also not chosen by ν' in $\nu'(\widehat{\mathscr{P}'})$. Note that the case where x does not occur in $\nu(\widehat{\mathscr{P}})$ is trivial and it corresponds to a scenario in which $fie(x)$ maps to Expression 1 in Lemma 4.1 with $\mathscr{F}_s = \mathscr{F}_1^+ \cap \mathscr{A}_s$, $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$ and $\mathscr{F}_2 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$. Now, let $x \notin \nu(\widehat{\mathscr{P}'})$ and $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$. That is, there is an insertion u in Δ that adds the node x as a child of a node y in $\nu(\widehat{\mathscr{P}'})$. Let $\mathscr{F}_2^+ = (\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_2})$ be the set of all valid ancestor events of e_2 in $\mathscr{F}$. By the definition of Δ, we can state that the formula $fie(x)$ of x in $\widehat{\mathscr{P}}$ maps to Expression 4 or 6 in Lemma 4.1 with $\mathscr{F}_s = \mathscr{F}_1^+ \cap \mathscr{A}_s$, $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$ and $\mathscr{F}_2 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$. If $fie(x)$ is Expression 4, this formula in $\widehat{\mathscr{P}'}$ is updated by Algorithm 1 which replaces e_1 in the first member (the conjunction) and e_2 in the second member by $(e_1 \vee e')$ and $(e_2 \vee e')$ respectively. Since ν' sets all the events in $\mathscr{F}$ to true, the first member of $fie(x)$ will be valuated to false. As for the second member, it will be valuated to true under ν' because all the events in $\mathscr{F}_2$ are set to true and $(e_2 \vee e') \models \nu'$. As a result, ν'

[6] The last relation is due by the fact that the events e_1 and e_2 are both valuated to true.

will select x in $\nu'(\widehat{\mathscr{P}'})$ since $\mathit{fie}(x)$ is such that $\mathit{fie}(x) \models \nu'$. Otherwise, if $\mathit{fie}(x)$ is Expression 6, it is updated in $\widehat{\mathscr{P}'}$ by `mergePrXML` which substitutes e_2 by $(e_2 \vee e')$. Given that it is straightforward to prove that $\mathit{fie}(x) \models \nu'$ since all the events in $\mathscr{F}_2$ are set to `true` and $(e_2 \vee e') \models \nu'$.

Scenarios (a) and (b) demonstrate that when $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$, then $x \in \nu'(\widehat{\mathscr{P}'})$. Similarly, the converse can be reached. We conclude $[\nu(\widehat{\mathscr{P}'})]^{\Delta} = \nu'(\widehat{\mathscr{P}'})$ which joint with $\nu(\widehat{\mathscr{P}'}) = \Omega(\mathscr{F}_1^+)$ and $\nu'(\widehat{\mathscr{P}'}) = \Omega''(\mathscr{F})$ yield $[\Omega(\mathscr{F}_1^+)]^{\Delta} = \Omega''(\mathscr{F})$. At last, the result $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ relies on $\Omega(\mathscr{F}) = [\Omega(\mathscr{F}_1^+)]^{\Delta}$.

4. For each subset $\mathscr{F}$ such that $\{e_2, e'\} \cap \mathscr{F} \neq \emptyset$ and $e_1 \notin \mathscr{F}$, we have $\Omega'(\mathscr{F}) = [\Omega((\mathscr{F} \setminus \{e'\}) \cap (\mathscr{A}_{e_2} \cup \{e_2\}))]^{\Delta_1 - \Delta^{\mathscr{C}}}$. This scenario is entirely symmetric to Scenario 3.

5. For each subset $\mathscr{F}$ such that $\{e_1, e_2\} \cap \mathscr{F} = \emptyset$ and $e' \in \mathscr{F}$, we have $\Omega'(\mathscr{F}) = [\Omega((\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)))]^{\Delta_3 - \Delta^{\mathscr{C}}}$. Let set $\mathscr{F}' = (\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))$. Given a valuation ν setting all the events in $\mathscr{F}'$ to `true` and the revision variables in all other events to `false`, we can write $\Omega(\mathscr{F}') = \nu(\widehat{\mathscr{P}})$ and $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$ according to Scenarios 1 and 2. Similarly to Scenario 3, we have now to demonstrate that $[\nu(\widehat{\mathscr{P}'})]^{\Delta_3 - \Delta^{\mathscr{C}}} = \nu'(\widehat{\mathscr{P}'})$ (where ν' is the valuation setting all the events in $\mathscr{F}$ to `true` and the revision variables of all the remaining events to `false`) to obtain that $[\Omega(\mathscr{F}')]^{\Delta_3 - \Delta^{\mathscr{C}}} = \nu'(\widehat{\mathscr{P}'})$. This result will be sufficient for stating that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ since by definition $\Omega'(\mathscr{F}) = [\Omega(\mathscr{F}')]^{\Delta_3 - \Delta^{\mathscr{C}}}$ and $\nu'(\widehat{\mathscr{P}'}) = \Omega''(\mathscr{F})$. Let us consider $\Delta = \Delta_3 - \Delta^{\mathscr{C}}$. For the proof, the intuition here is to see that by implementation Δ can be rewritten as $(\Delta_1 - \Delta^{\mathscr{C}}) \cup (\Delta_2 - \Delta^{\mathscr{C}})$ on one side. So, if we arrive to show that for all the nodes x, y such that $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta_1 - \Delta^{\mathscr{C}}}$ and $y \in [\nu(\widehat{\mathscr{P}'})]^{\Delta_2 - \Delta^{\mathscr{C}}}$, then $x \in \nu'(\widehat{\mathscr{P}'})$ and $y \in \nu'(\widehat{\mathscr{P}'})$, we can trivially deduct that for each node $x \in [\nu(\widehat{\mathscr{P}})]^{\Delta}$, then $x \in \nu'(\widehat{\mathscr{P}'})$. These relations can be proven in the same spirit as in Scenario 3 by just noting that:

(a) For the set of unmodified nodes x in $\nu(\widehat{\mathscr{P}'})$ with Δ, $\mathit{fie}(x)$ in $\widehat{\mathscr{P}}$ is compatible to Expression 1 in Lemma 4.1 with $\mathscr{F}_1 = \mathscr{F}' \cap \mathscr{A}_s$, $\mathscr{F}_1 = (\mathscr{F} \cap (\mathscr{A}_{e_1} \setminus \mathscr{A}_s)) \cup \{e_1\}$, and $\mathscr{F}_1 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$.

(b) For the set of nodes x in $\nu(\widehat{\mathscr{P}'})$ handled with Δ. If the operation comes from $(\Delta_1 - \Delta^{\mathscr{C}})$, $\mathit{fie}(x)$ is compatible to Expression 3 or 5 in case of insertions and this formula maps to Expression 4 for deletions. Concerning operations in $(\Delta_2 - \Delta^{\mathscr{C}})$, $\mathit{fie}(x)$ is compatible to Expression 4 or 6 for insertions and to Expression 3 for deletions.

Furthermore, the inclusion in the opposite direction, that is for each node $x \in \nu'(\widehat{\mathscr{P}'})$, then $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$, is obvious for nodes unchanged by Δ. For the other nodes, we only need to verify that they are correctly handled by Δ in $\nu(\widehat{\mathscr{P}'})$ depending whether it corresponds to an addition or a deletion in this document. Following that, we can state that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ holds for each subset $\mathscr{F}$ of events in $\mathscr{V}$ that contains e' but not e_1 and e_2. □

We conclude by showing efficiency of `mergePrXML`.

PROPOSITION 4.4. `mergePrXML` *performs the merge over the encoding of any uncertain multi-version XML document in time proportional to the size of the formulas of nodes impacted by the updates in merged branches.*

PROOF. (Sketch) The intuition behind the time complexity is that `mergePrXML` results in a constant-time update of DAG $\mathscr{G}$, firstly. Secondly, by viewing $\widehat{\mathscr{P}}$ as an amortized hash table, the algorithm retrieves any node impacted by a (non-conflicting) update in constant time. Finally, the `replace` method only depends on the lengths of formulas. □

5. CONCLUSION

We presented in this paper a merge operation, as well as an efficient mapping algorithm, that supplements our uncertain multi-version XML framework presented in [2]. The merging mechanism proposed covers common deterministic XML merge scenarios while managing uncertain data. Example applications include the merging of documents in uncertain version control platforms such as Wikipedia.

6. ACKNOWLEDGEMENTS

This work was partially supported by the Île-de-France regional DROD project, and the French government under the STIC-Asia program, CCIPX project. We would like to thank the anonymous reviewers for their valuable suggestions on improving this paper.

7. REFERENCES

[1] T. Abdessalem, M. L. Ba, and P. Senellart. A probabilistic XML merging tool. In *Proc. EDBT*, 2011.

[2] M. L. Ba, T. Abdessalem, and P. Senellart. Uncertain version control in open collaborative editing of tree-structured documents. In *Proc. DocEng*, 2013.

[3] G. Cobena, T. Abdessalem, and Y. Hinnach. A comparative study for XML change detection. In *BDA*, 2002.

[4] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version Control with Subversion*. O'Reilly Media, 2008.

[5] J. Estublier. Software configuration management: A Roadmap. In *Proc. ICSE*, 2000.

[6] R. L. Fontaine. Merging XML files: A new approach providing intelligent merge of XML data sets. In *Proc. XML Europe*, 2002.

[7] S. Khanna, K. Kunal, and B. C. Pierce. A formal investigation of Diff3. In *Proc. FSTTCS*, 2007.

[8] E. Kharlamov, W. Nutt, and P. Senellart. Updating Probabilistic XML. In *Proc. Updates in XML*, 2010.

[9] B. Kimelfeld and P. Senellart. Probabilistic XML: Models and Complexity. In *Advances in Probabilistic Databases for Uncertain Information Management*. Springer-Verlag, 2013.

[10] T. Lindholm. A three-way merge for XML documents. In *Proc. DocEng*, 2004.

[11] J. Ma, W. Liu, A. Hunter, and W. Zhang. An XML Based Framework for Merging Incomplete and Inconsistent Statistical Information from Clinical Trials. In Z. Ma and L. Yan, editors, *Software Computing in XML Data Management*. Springer-Verlag, 2010.

[12] L. Peters. Change detection in XML trees: a survey. In *TSIT Conference*, 2005.

[13] N. Suzuki. A Structural Merging Algorithm for XML Documents. In *Proc. ICWI*, 2002.